ICL PERQ

Guide to PNX

ICL endeavours to ensure that the information in this document is correct, but does not accept liability for any error or omission.

Any procedures described in this document for operating ICL equipment should be read and understood by the operator before the equipment is used. To ensure that ICL equipment functions without risk to safety or health, such procedures should be strictly observed by the operator.

The development of ICL products and services is continuous and published information may not be up to date. Any particular issue of a product may contain part only of the facilities described in this document or may contain facilities not described here. It is important to check the current position with ICL.

Specifications and statements as to performance in this document are ICL estimates intended for general guidance. They may require adjustment in particular circumstances and are therefore not formal offers or undertakings.

Statements in this document are not part of a contract or program product licence save insofar as they are incorporated into a contract or licence by express reference. Issue of this document does not entitle the recipient to access to or use of the products described, and such access or use may be subject to separate contracts or licences.

R10139/00

Second Edition January 1984

ICL will be pleased to receive readers' views on the contents and organisation, etc. of this publication. Please write to:

The Registry (Readership Survey)
UK Software and Literature Supply Sector
International Computers Limited
60 Portman Road
Reading
Berks RG3 1NR

Distributed by UK Software and Literature Supply Sector
International Computers Limited
Registered Office: ICL House, Putney, London SW15 1SW
Printed by ICL Printing Services
Engineering Training Centre, Icknield Way West
Letchworth, Herts SG6 4AS

Preface

This publication provides a detailed reference guide to release 2.0 of thePNX operating system on the ICL PERQ. PNX is an ICL version of the UNIX operating system designed to run on both PERQ 1 (PERQs with the 14 inch fixed disc, that have been fitted with the enhanced I/O firmware option) and PERQ 2 (PERQs with an 8 inch fixed disc). Because PNX consists mainly of standard UNIX software, this *Guide to PNX* must be read in conjunction with:

UNIX Programmer's Manual, Volume 1 (R 10125/00, Edition 7)
UNIX Programmer's Manual, Volume 2A (R 10126/00, Edition 7)
UNIX Programmer's Manual, Volume 2B (R 10127/00, Edition 7)

which are exact copies of the Seventh Edition of the Bell Laboratories publications of the same name, made available through ICL, by agreement, for your convenience. This publication is only a supplement to existing UNIX literature so if you are not already familiar with UNIX you should follow the documentation guide supplied with this publication (see Appendix 1).

If you are not already familiar with the ICL PERQ, you should also read:

ICL PERQ: User Guide (R10119/00, Preliminary Edition) if you have a PERQ1 or

ICL PERQ: User Guide (R10132/00, Edition 1) if you have a PERQ 2

for advice on using the hardware.

Further publications describe the optional printers:

ICL 3185 Matrix Printer on PERQ (R10135/00, Edition 1)

ICL 6202/02 Correspondence Printer on PERQ (R10136/00, Edition 1)

ICL 6202/03 Correspondence Printer on PERQ (R10137/00, Edition 1)

ICL 6203 Electrostatic Printer/Plotter on PERQ (R10138/00, Edition 1)

Details of the procedure for installing an ICL PERQ system are given in the publications:

ICL PERQ: Installing your ICL PERQ (Edition 2, TP10102) for PERQ 1

ICL PERQ: Installing your PERQ (Edition 1, 5156224) for PERQ 2

The installation card accompanies the relevant PERQ hardware. Details of the actions needed to install the PNX operating system on the ICL PERQ are covered in the Software Release Notice delivered with the software set.

PNX currently supports three programming languages; Pascal, C and UNIX FORTRAN 77. C and UNIX FORTRAN 77 are part of the standard PNX, Pascal is a separate product.

If you intend to write Pascal programs while running PNX, in addition to this *Guide to PNX* you need the following publications:

ICL PERQ: Developing ICL Pascal programs under PNX (R10130/00, Edition 1)

ICL-Pascal Language (R02269/00, Edition 1)

If you intend to write C programs while running PNX, in addition to this *Guide to PNX* and the *Unix Programmer's Manual* you need the standard text on C:

The C programming Language, Kernighan, B.W. and Ritchie, D.M., Prentice Hall, Englewood Cliffs, New Jersey, 1978, ISBN 0-13-110163-3

If you intend to write UNIX FORTRAN 77 programs while running PNX, in addition to this *Guide to PNX* you need the ICL publication:

FORTRAN 77 Language (R02254/00, Edition 1)

PNX provides several communications facilities, the major ones being covered in detail in separate publications:

ICL PERQ: 2780/3780 Communications under PNX (R10133/00, Edition 1)

ICL PERQ: Establishing IPA under PNX (R10131/00, Edition 1)

The Bell Laboratories publication:

UNIX Reference Card

is also substantially relevant to PNX on PERQ.

This publication is structured as follows:

Chapter 1 is an introduction to the PERQ and the PNX system and highlights the differences between PNX and standard UNIX.

Chapter 2 covers setting up PNX and is an introduction to using the system and system management.

Chapter 3 describes the Window Management System.

Chapter 4 describes *spy*, the PNX text editor, and provides a tutorialexercise for those new to *spy*.

Chapter 5 describes using cursors and using *cedra*, a cursor editor. This chapter also includes programming examples.

Chapter 6 describes the special files in PNX corresponding to Input/Output devices and the facilities provided by PNX to drive these devices.

Chapter 7 is an overview of the communicatiqns potential of a PERQ running PNX providing instructions for simple use of *chatter*, *uucp* and *mftp*.

Chapter 8 covers program development and mixed programming.

Appendix 1 is a documentation guide for those unfamiliar with UNIX and includes a suggested scheme of reading.

Appendix 2 lists all the components of UNIX (Seventh Version) and indicates those that are not included in PNX. This appendix also lists all the additions and modifications provided by PNX, thus providing a full contents list for the components of PNX.

Appendix 3 is a collection of all the amended and additional components to UNIX in the style of the *UNIX Programmer's Manual, Volume 1 (UPM(1))* for insertion into the *UPM(1)*.

Appendix 4 describes developing UNIX FORTRAN 77 programs under PNX.

Appendix 5 describes developing C programs under PNX.

Appendix 6 shows the PERQ ASCII character set and the raw keyboard values.

Appendix 7 list the diagnostic display codes relevant to faults users can fix.

Appendix 8 contains articles that describe the System 3 facilities SCCS and *fsck*(1) which have been added to PNX. The articles are copies from the *UPM* for System 3 UNIX.

Contents

The text of this publication is divided into chapters in the normal way, and each chapter is subdivided into sections. A section's level in the hierarchy is indicated by its number. Therefore, within Chapter *n*, first level section headings are numbered *n*.1, *n*.2 and so on; second level headings are numbered *n*.1.1, *n*.1.2 ... *n*.2.1 and so on; third level headings are numbered *n*.1.1.1, *n*.1.1.2 ... *n*.1.2.1 and so on.

The contents list and index, and cross-references in the text, all refer to section numbers.

Pages are numbered within chapters, in the form *c-p*, where *c* is the chapter number and *p* the page number within that chapter. Figures and tables, where they appear, are also numbered within chapters, so that Figure *n*.2 is the second figure in Chapter *n*, and Table *n*.2 is the second table in that chapter.

Section numbers, page numbers and figure and table numbers in appendices are preceded by the letter A.

This chapter introduces you to both the PERQ itself and to PNX, an operating system developed from the UNIX operating system but tailored specifically for the PERQ. Chapter 2 introduces you to setting up and using the PNX operating system.

1.1 The PERQ environment

PERQ is a single user computer of great power, having a large internal data storage capacity and a very fast processor. Additionally, PERQ has a large storage capacity for permanent memory with its Winchester type fixed disc and the floppy disc drive allows you to exchange programs and data with other users, receive software from your supplier and keep a backup file system. However, the important feature of the PERQ is the way that it communicates; with you, with instruments and with other computers.

Currently there are two types of PERQ that support PNX. PERQ 1 is the original PERQ (with a 14 inch Winchester type fixed disc) fitted with the enhanced I/O firmware option. PERQ 2 designates the new PERQ hardware characterised by an 8 inch Winchester type fixed disc. PERQ 1 and PERQ 2 have the following features:

* An A4 size high resolution graphics display screen. The 1024 x 768 bit mapped raster display with its 60 Hz refresh rate allows the presentation of extremely detailed diagrams, many different fonts for text, and real animation. Additional hardware instructions, RasterOp, Line and DrawByte, enable all or part of the display to be changed by a single instruction
* A standard typewriter style keyboard with extra function keys provided on PERQ 2 for text and command input
* A graphics tablet with three button pointing device (the puck) which can be used to input or construct diagrams or point to parts of the display. An optional high resolution graphics tablet with a four button pointing device is also available (this is the standard tablet for PERQ 1)
* Two RS232C interfaces (one on PERQ 1) for communication with serial devices such as some printers or for communication on wide area networks to other devices and computers
* A GPIB interface (IEEE - 488 standard) for communicating with parallel devices such as laboratory instruments and some printers
* An optional Open Systems LAN interface
* A Z80 microprocessor dedicated to Input/Output (I/O)
* Speech hardware

Detailed descriptions of all the PERQ hardware components are in the publications *ICL PERQ: User Guide.*

1.2 The PNX environment

UNIX is an operating system originally produced by Bell for program development in a research environment. It is now extremely popular throughout the world, particularly in universities, and is accepted as an operating system for supporting applications in other environments as well. The reason for its popularity is its simplicity and generality. Most of UNIX and the software that runs under it is written in C. Very little of the operating system is machine dependent. This means that UNIX software is highly portable. The most important feature of UNIX is the speed at which complex applications can be produced by connecting together simpler existing programs.

PNX is implemented on the PERQ by microcoding so that the apparent machine code is C-code. C-code is tailored to the needs of the C language since the majority of PNX software consists of unchanged, standard UNIX software written in C. PNX on the PERQ has the well known features of the UNIX operating system plus the advantage of running on a powerful graphics oriented computer.

If you are unfamiliar with standard UNIX, Appendix 1 provides a useful guideto the available UNIX documentation. Because PNX consists mainly of standard UNIX software, your main reference documentation for PNX is the *UNIX Programmer's Manual(UPM)*. Volume 1 of the *UPM*, hereafter referred to as *UPM*(1), contains specifications for all the UNIX utilities. Volumes 2A and 2B, hereafter referred to as *UPM*(2A) and *UPM*(2B), contain articles describing various parts of the UNIX system. Appendix 2 indicates which UNIX facilities are not provided under PNX. This *Guide to PNX* is a supplement to the *UPM* and documents the additions to standard UNIX provided by PNX to take advantage of the PERQ's capabilities. PNX provides the following additions:

1. WINDOW MANAGEMENT SYSTEM This exploits the high resolution graphics display and enables you to take full advantage of the multiprocessing ability of PNX. The Window Management System also includes a pop up menu package so you can easily incorporate pop up menus into your programs. Chapter 3 describes using the Window Management System (WMS) and the pop up menu package

2 SCREEN EDITOR This point and act editor uses the graphics tablet and puck as the main user interface, making text editing quick and easy. Chapter 4 describes using the screen editor

3 CURSOR EDITOR You can design the cursor which indicates the position and effect of the puck. Chapter 5 describes programming and designing cursors

4 DEVICE DRIVERS These enable you to use standard UNIX calls to drive PERQ hardware and the optional devices and printers which may be attached to the RS232C and GPIB interfaces. The I/O devices and optional printers supported under PNX and the related device drivers are described in Chapter 6

5 VIRTUAL STORE This supplements the UNIX RIRO system making it possibleto run programs larger than the PERQ real main store. Virtual store is described in section 1.2.1

6 COMMUNICATIONS As well as the standard UNIX *uucp* facility, PNX supports five more communications facilities including an IBM 2780/3780 emulation which is available as a separate product. PNX provides two communications interfaces; an RS232C port (two on PERQ 2) and an Open Systems LAN port (compatible with Ethernet). Chapter 7 provides an overview of the PNX communications potential

7 SYSTEM 3 ADDITIONS PNX includes some System 3 UNIX utilities and libraries. The System 3 components are additions to the system and do not replace existing Version 7 UNIX components. In particular, PNX provides SCCS, the Source Code Control System. This is a set of commands to control and account for changes to files of text, particularly source code. PNX also provides *fsck*(1). For details of SCCS and *fsck*(1) see Appendix 8

8 USER MICRODING This facility is available as a separate product

Other changes to standard UNIX are as follows:

1 LANGUAGES PNX provides FORTRAN 77 and C and Pascal is available as a separate product. Chapter 8 supplies information on program development under PNX. Appendix 4 describes the features of FORTRAN 77 relevant to program development, Appendix 5 describes the features of C and Pascal is described in the publication *ICL PERQ: Developing Pascal programs under PNX*

2 ASSEMBLER SOFTWARE UNIX items written in PDP assembler have either been rewritten for the PERQ or omitted if they are not relevant

3 NON-PERQ HARDWARE SPECIAL SOFTWARE All such items have been omitted

1.2.1 Virtual store

PNX retains the RIRO system, but supplements it with a paging system, which handles the three store areas in any process in the following way:

1 STACK While a process is rolled in, its whole stack is held in main store, that is, it is not paged

2 OFF-STACK DATA While a process is rolled in, its first 128Kb page of off-stack data is always held in store, and the next seven 128Kb pages are brought in and written out as required by the process and by the amount of available store. A similar system is used for any subsequent pages, but the mechanism employed is not as fast

3 CODE The size of a code page (*text* page in UNIX terminology) may vary between different processes although all the pages of a process are the same size. The page size is the multiple of 512 bytes that can contain the largest procedure in the process with a minimum size of 8Kb. While a process is rolled in and running, at least one page (the currently active one) is held in main store

Any one process must fit within the following limits:

Item	*Limit*
Off-stack data	4Mb
Stack	512Kb
Code	4Mb

The size of the swapfile imposes a total limit of 6Mb on the size of programs which are multiprogrammed.

As long as processes fit within the above limits the PNX system is able to run them, although a process with off-stack data greater than 1Mb (more than 8 pages) may run rather slowly and should be run with the smallest possible number of other processes.

1.3 Typographical conventions

As this publication is a supplement to the *UPM,* the manual follows the *UPM* typographical conventions.

Specifications that appear in the *UPM*(1) or in Appendix 3 of this publication are written in lower case italics with a number in brackets to indicate to which section of the UPM they belong, for example *spy*(1), *open*(2). In places these specifications are also referred to by their pathnames for example, */hd/etc/passwd.*

In example command lines, commands that should be typed exactly as they appear are printed in bold. Generic terms, for example, *filename* are printed in italics.

Optional parameters are enclosed in square brackets.

Ellipses indicate that an argument type may be repeated.

Actual keys that should be pressed are also printed in bold, for example **RETURN**. Control keys, that is, where **CONTROL** and, for example, **Z** must be pressed simultaneously, are indicated by the ^ symbol preceding the key, for example, **^Z**.

2.1 Installing PNX

PNX is supplied as a number of floppy discs, three of which are boot floppy discs, one for each hardware type, and the rest are filestore floppy discs. A release notice accompanying the software set gives a list of floppy discs making up the software set and full details of how to set up PNX. Make sure you use the right boot floppy disc.

If you are moving from another system read the instructions below before starting to set up PNX:

1 Moving from POS

Before you install PNX you should preserve any POS files on your fixed disc by copying them (with the POS FLOPPY utility) to floppy discs. *Once you boot from the first PNX boot floppy disc you will lose any information on the fixed disc.*

When PNX is installed you can use the *fl*(1) utility to transfer the POS files into the PNX file system on the fixed disc. Note that POS floppy discs do not look like PNX file systems, so they cannot be mounted, and *fl*(1) is the only way you can read from or write to them

2 Moving from an older PNX

Follow the instructions for setting up PNX but instead of giving the command *makepnx*(1) give the command *updatepnx*(see *makepnx*(1))

If having set up PNX you lose some of the files you may need to access particular files on the floppy discs. The labels on the floppy discs do not completely reflect their contents. To find out where a particular file is, use *mountflop*(1) to mount the disc and *ls*(1) to list the contents.

2.1.1 Loading PNX

Starting up

Remove any floppy disc from the drive before switching on the PERQ. For instructions on switching on the PERQ see the publication *ICL PERQ: User Guide.* The screen warms up within a few seconds of you switching it on. Turn the brightness control at the back of the display so that you can see an uncontrolled flickering grey background with several white lines tracing across the screen. The fixed disc takes about 30 seconds to spin up to speed (about 90 seconds for PERQ 1) so you should leave it for this time before worrying about trying to boot PNX.

If there is no floppy disc in the drive, and the fixed disc has software on it, PERQ 1 tries to boot from the fixed disc as soon as it is able. Normally, this is just what you want, but when installing PNX you must boot from the special boot floppy provided instead; if PERQ 1 does start booting from the fixed disc just ignore it and carry on as detailed below.

Booting

When you switch on a PERQ, its processor contains no software at all. The only action it can perform is to load a small amount of software by reading predefined blocks from either the fixed disc or a floppy disc. This software is then executed and its main function is to load more software from the disc until the operating system is fully loaded and ready to use. This process has become known as bootstrapping or *booting.* As booting proceeds, a three digit LED display (under the keyboard for PERQ 1, on the processor box for PERQ 2) displays diagnostic codes to show each stage of the booting process. If booting fails, the code indicates the cause of the failure. Early codes refer to hardware problems. Appendix 7 lists all the codes relevant to faults users can solve. For other faults, call your ICL service engineer quoting the displayed code.

To carry out the initial boot follow this procedure:

1 Put the PNX boot floppy disc in the drive and press and release the boot switch. Wait about 60 seconds for the login display to appear

2 Type in the date and time as requested. You are now in multiuser mode

3 Log in with a username of *root* and a password of *root*. The password is not echoed to the screen. You are now logged on as the superuser

4 When you get the # command prompt, issue the *makepnx*(1M) or *updatepnx* command. This initialises the filestore on the fixed disc and copies some files from the boot floppy to the fixed disc. Nothing happens for several minutes while empty files are built onto the fixed disc but after a while the floppy drive will start to clunk as files are copied from it to the fixed disc. After about 20 minutes the command prompt, #, reappears

5 Issue the *bye*(1) command

6 Wait for the red light on the floppy drive to go out before removing the floppy disc

Loading the filestore floppy discs

1 Now reboot the system from the fixed disc by pressing the boot switch

2 Log in as before and for each floppy disc in the software set load the floppy disc in the floppy drive and issue the *prime*(1) command. The # command prompt reappears to show you when each command is finished

In total, PNX takes about an hour and a quarter to load. It does not matter which order you *prime*(1) most of the discs but it is better to load them in the order suggested in the Release Notice.

2.1.2 Setting up the system

Most of the things needed to set up the system are system management functions covered in section 2.3, for example, setting up usernames and user directories, setting up the file and setting up network and peripheral connections. It is also possible for you to program in certain features of PNX.

At every boot the program *init*(8) is run. *init*(8) brings up the single user mode and multiuser mode login displays. When *init*(8) comes up multiuser, it invokes a shell with input taken from */etc/rc*. */etc/rc* is a command file containing general command lines understood by the shell and you can put entries into */etc/rc* to set up PNX the way you want it.

/etc/rc already contains entries when you first install PNX. One of the entries disables the microcode debugger (see *debugoff*(1)).

Make entries in */etc/rc* as follows:

1 Make sure you are still logged on with the # command prompt (that is, you are the superuser)

2 Type the command:

ed /etc/rc

The editor tells you the current size of the file.

3 Type the following lines:

a
command line
.
w
q

where *command line* represents as many command lines as you wish to enter.

The supplied system is set up to use the Summagraphics Bit Pad tablet. In order to use the Kriz tablet issue the following two command lines:

rm /dev/tablet

/etc/mknod /dev/tablet c 12 0

6 300 B.P. 82

2.2 Using PNX

2.2.1 Daily switching on

1 Set the power switch on the processor cabinet to ON. The screen display flickers and you should hear the fans start. If you do not hear the fans start, consult the publication *ICL PERQ: User Guide*

2 For PERQ 2, press the boot switch. Booting should occur automatically on PERQ 1 but if after two minutes PERQ 1 has not booted, press the boot switch

Once loaded PNX automatically runs *fsck*(1), a file system consistency checking program. *fsck*(1) displays status messages to show you what it is checking and, if it comes across an error, prompts you before taking any action. To maintain performance, it is necessary that *fsck*(1) be run fairly frequently. *fsck*(1) has been included in the initial boot process to ensure that this happens. For details on running *fsck*(1) see section 2.3.3.3.

The multiuser mode login display appears automatically once *fsck* has run, unless *fsck* detected errors or aborted. If *fsck* finds an error the single user login prompt appears.

2.2.2 Logging in

PERQ under PNX may be used in one of two modes, single user mode or multiuser mode. Whenever you switch on the PERQ, *fsck* is run and as long as no errors occur the machine automatically enters multiuser mode and starts the multiuser login display.

First you are asked to enter the date and time. Once you have entered the date and time the login prompt appears:

login:

PNX expects you to enter a *username.* Enter your username followed by your password when prompted. The password you type is not echoed to the screen. See the following sections for details on getting a username and password.

If PNX is running WMS, you can get the login display in a particular window by typing the command:

login

Log in as usual in response to the login prompts. When windows are created they assume the log in name in use when *winit* initiated the WMS. Any **.profile** file belonging to that user is run in every window that is created or opened while WMS is running. If you then log in to a window, the log in name changes for that window only and any processes run subsequent to the log in assume the new log in name for access permissions.

You should always use the PERQ in multiuser mode unless performing some system management (see section 2.3). You can enter single user mode by pressing ^Z in response to the shell command prompt (**#** if you are superuser, **$** if you are an ordinary user). If *fsck* finds an error then the machine enters single user login mode not multiuser login mode. The single user login prompt does not first ask for the date and time but displays the login prompt straight away.

You only need to be in single user mode to run *fsck* in order to correct file system inconsistencies (see section 2.3.3.3).

If in single user mode, typing ^Z in rsponse to the login prompt causes the machine to enter multiuser mode.

2.2.2.1 *Getting a username*

PNX maintains a file *passwd*(5) which lists all the usernames that PNX recognises. To log in to PNX successfully you need to supply a username, and possibly a password, that PNX recognises.

Your system manager may supply you with a username. If not, follow the instructions in section 2.3.2.

2.2.2.2 *Establishing a password*

The line entry in *passwd*(5) associates a numerical user ID, a group ID and a home directory with each username.

When you create a file it is marked with your user and group ID. Each file is also marked with a set of protection bits specifying read, write and execute permissions for the owner of the file, members of the same group and remaining users respectively.

When you log in with a particular username you are constrained by the permissions on a file. PNX maintains passwords to prevent users logging in under someone else's username. To establish or change a password follow this procedure:

1 Issue the command **passwd** (see *passwd*(1)) with your username as a parameter

2 If you already have a password the program prompts you for the old password. Type the old password. This is not echoed to the screen

3 The program prompts you for the new password. Type the new password.This is not echoed to the screen

4 The program prompts you for the new password a second time to confirm that you have typed it correctly. Type the new password again

The password needs to be at least six characters long if all in one case.

2.2.2.3 **.profile** files

The entry in *passwd*(5) for each recognised user names a home or working directory for the user. When you log in the shell looks in */etc/passwd* and makes your home directory the current directory.

The shell also looks in the home directory for a **.profile** file and, if it finds one, executes any commands contained in the file.

A **.profile** file is a file, set up like any other text file, containing command lines for the shell to execute before doing anything else. The file should be named with the suffix **.profile** and entered in the home directory.

Useful **.profile** commands would be *pwd*(1), which prints the working directory, and *ls*(1), which lists the contents of the directory.

2.2.3 Entering commands

Once you have logged in successfully, PNX displays the standard command prompt:

$

inviting you to enter commands.

If you log in as the superuser (login name *root*) the standard command prompt is:

#

2.2.3.1 *Typing commands*

PNX recognises the commands listed in Section 1 of the *UPM*(1) and the name of any executable program in the current directory.

To enter commands follow these rules:

1 Type all commands in lower case

2 Separate commands, options and parameters by spaces. The synopsis section of each command specification summarises the appearance of a command line

3 Edit mistyped commands using **DEL** or **OOPS**, not **BACKSPACE**

The shell recognises a certain number of metacharacters in filename parameters (see *sh*(1)).

2.2.3.2 *Puck commands*

Some PNX programs, for example, Window Manager, *spy*(1), provide a menu of commands. Use the puck to select commands as follows:

1 Move the puck so the hardware cursor on the screen points at the required command. The appearance of the hardware cursor depends on the program

2 Press the button to select the command. The button to press depends on *wprofile*(5) but the default is *left* on the three button puck and *yellow* on the four button puck. The selected command becomes highlighted in inverse video (that is, what is white becomes black and what is black becomes white)

3 Release the button to effect the selection. While the button is pressed you can select any other command or abort by selecting outside the menu. It is the release that causes any action

Pop up menus disappear once you have made a choice or selected outside the menu.

Other puck commands depend on a particular button press in a particular area of the screen (see *spy*(1) or *cedra*(1)). Again it is the press that selects the command and the release that effects it.

2.2.3.3 *Control keys*

tty(4) describes the way in which key presses are interpreted. *wprofile*(5) enables you to set the Window Manager Escape key. *stty*(1) allows you to set *tty*(4) characteristics. The default control keys are as follows:

Window Manager Escape	**^RETURN**
Delete previous character	**DEL**
Delete all of current line	**OOPS**
Abort current command	**^C**

For full details see *tty*(4).

2.2.4 Filestore commands

When you log in, the shell places you in your home directory ready to begin work. Normally, all the files and directories you own would be within your home directory.

The UNIX file system is an hierarchical structure of directories. Directories do not actually contain files but contain an index table specifying the address of the files.

Figure 2.1 is a diagrammatic representation of the PNX file system showing how the directories are related and indicating the contents of each directory. *hier*(7) describes the standard UNIX file system in detail.

To access a file not in the current directory you need to specify the full pathname, for example, */usr/spool/lpd/file*. Alternatively you can change directory using *cd*(1). The . parameter to *cd* moves you into the parent directory of the current directory.

The following utilities are useful for moving through the PNX filestore to find what you want and manipulating files:

pwd(1)	print working directory	prints the name of the current directory
ls(1)	list	lists the contents of the current directory
cd(1)	change directory	allows you to specify a new working directory
mkdir(1)	make a directory	creates a directory within the current working directory
cat(1)	catenate and print	prints a file to the standard output, usually the screen

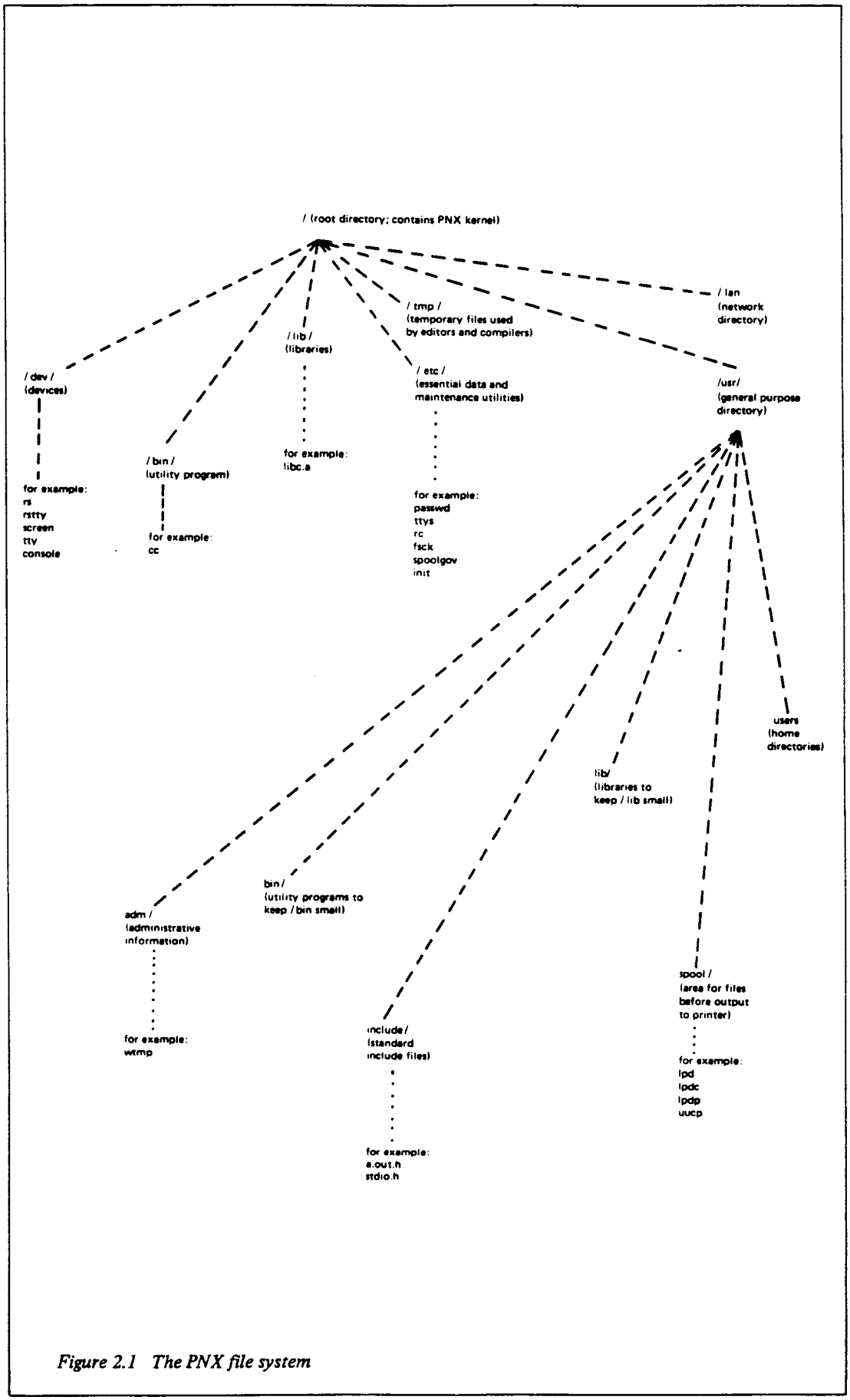


Figure 2.1 The PNX file system

cp(1)	copy	copies a file from any directory to another
mv(1)	move	moves a file from any directory to another
spy(1)	interactive text editor	edits or creates a file
ed(1)	UNIX text editor	edits or creates a file
find(1)		finds a file
rmdir (see *rm*(1))	remove directories or files	removes the entry for a file in a directory. If the file is not an entry in any other directories it is destroyed
dircmp(1)	compare directories	lists entries that are unique to each or common to both
grep(1)	search a file for a pattern	copies the line containing the pattern to the standard input

2.2.5 Running programs

Having created a source file using one of the PNX editors, you need to compile and assemble the program into an object file.

FORTRAN 77 and C compilers are provided as part of the standard PNX product set. A Pascal compiler is also available as a separate product. The compilers are:

cc(1)	for C source files
f77(1)	for FORTRAN 77 source files
ipc(1)	for Pascal source files (see *ICL PERQ: Developing Pascal programs under PNX)*

Having compiled your source files into object files using one of the above compilers you then need to combine the files to create an executable program.

ld(1) combines several object programs into one, resolves external references and searches libraries to create an executable program called **a.out.** An option to *ld*(1) enables you to provide an alternative name for this output file.

To run an executable program just type the name of the output file.

2.2.6 Logging out

If WMS is running, quit WMS by deleting the system window and return to the shell.

Typing **^Z** returns you to the single user login prompt.

If you are superuser, typing **shutdown**(*shutdown*(1M)) returns you to the single user login display.

To log out preparatory to switching off, follow one of two procedures:

1 In multiuser mode type:

bye

This utility performs a shutdown, performing all writes to the superblock, and parks the heads at the centre of the fixed disc. This is essential if you intend moving the PERQ. For details see *bye*(1)

2 In single user mode type:

kill -1 1
sync

The *kill* signal halts all processes and the sync performs all necessary writes to the superblock

If these instructions are not followed, the superblock may not be updated, causing data loss.

2.2.7 Switching off

Having logged out, set the switch on the processor cabinet to OFF. Remember to switch off all peripherals.

2.3 System management

The UNIX operating system was initially developed for multiuser systems. Such systems normally have a dedicated systems manager and perhaps some operations staff. On the single user PERQ system, however, it is the user's responsibility to carry out system management functions, that is, user management, filestore maintenance and network management.

You can only perform the system management functions if you are logged in as the *superuser.*

When PNX is installed it sets up a few login names; *root, bin, daemon, sys, demo* and *guest. root* is the login name for the *superuser.*

2.3.1 The superuser

If you log in with the superuser login name *root* you are all powerful. All file protection is lifted and the system can easily be destroyed.

Initially the password associated with *root* is also *root.* As the system manager, the first thing you should do is change the superuser password so that only you can log in as the superuser (see section 2.2.2.2).

The boot floppy disc used to set up PNX has its own *passwd*(5) file. You must keep that disc secure. If by any chance you forget the superuser password follow this procedure:

1 Boot from the floppy disc

2 Login as the superuser using the login name *root* and the old superuser password *root*

3 Mount the fixed disc using:

/etc/mount /dev/hard /hd

4 Edit the file */hd/etc/passwd* to set the superuser password to null

5 Issue the *sync* command

6 Reboot from the fixed disc

7 Set up a new superuser password

2.3.2 User management

Although the PERQ is a single user machine it may be used by several people at different times and with the communications potential may well be used by more than one user at the same time. Each user has a login name and their own set of files associated with that login name. Users may further protect their own files with a password.

2.3.2.1 *Maintaining passwd*(5)

One of the tasks of the superuser is to add and remove users from the file *passwd*(5). Only the superuser can do this.

passwd(5) describes the structure of the file. */etc/passwd* consists of line entries for each recognised user. Each line entry contains seven fields separated by colons.

To add a new user you need to supply a line entry as described in *passwd*(5) and create the home directory.

To remove a user, delete the line entry in */etc/passwd.* This does not remove the user's home directory and all the files contained in it. These have to be removed using *rmdir*(1).

Do not try to remove usernames supplied with the system, for example, root or you may find yourself with no way to access the file system.

2.3.2.2 *Changing a password*

To change another user's password, give the command:

passwd *username*

where *username* is the user's login name.

As the superuser you do not need to know the old password. Supply the new password when prompted. This is not echoed to the screen. You need to enter the new password twice. The superuser can create a null password.

2.3.2.3 *Accounting*

You can get the system to record how long each user is logged in by creating the log file */usr/adm/wtmp*. This file is maintained by *init*(8) and *login*(1) but is not created by them.

Every time a user logs in or out an entry is made in this file, which grows large very quickly. The command:

ac

summarises its contents by user name and date. You may keep a limit on the size of the file by adding a command as follows to the end of */etc/rc* (see *init*(8)):

test -f /usr/adm/wtmp &&
find /usr/adm/wtmp -size +50 -exec sh -c '>/usr/adm/wtmp'\;

It checks if the accounting file exists and if its size is over 50 blocks it empties it.

2.3.3 File management

2.3.3.1 *Backup files*

All users should copy important files onto floppy discs to safeguard against file system corruption. The superuser can create backup files but it is probably easier to allow each user floppy discs and make them responsible for their own files.

cptree(1) copies files from the fixed disc to the floppy disc. To copy your home directory follow this procedure:

1 Mount a blank formatted floppy which has had an empty PNX file system built on it. See *fl*(1), *mkflop*(1) and *mountflop*(1)

2 Issue the command:

cptree -a -d */usr/usename /fd/directoryname*

The command writes each file's name as it is copied

3 Unmount the floppy and remove it

The superuser can copy a user's directory preserving the original ownership of the files with the command:

cptree -a -d -t */usr/username /fd/directoryname*

To only copy files that have recent amendments use the **-m** option as follows:

cptree -a -d - M2a */usr/username /fd/directoryname*

where **-M2a** specifies files that have been amended in the last 2 days. You can specify the number of days and hours, for example, **-M1:3a** specifies files that have been amended in the last 1 day and 3 hours.

Files deleted from the user directory */usr/username* are not deleted on the backup floppy disc so the floppy disc will gradually fill up. At this stage you should reformat it, buid an empty PNX file system on it and use *cptree* to copy the whole directory again. It is a good idea to make two full backups onto two floppy discs and then alternate the update *cptree* commands between them.

Copying whole floppies

The *cptree* program is an adequate tool for copying PNX file systems between floppy disc and fixed disc, but to copy a floppy disc such as the PNX boot floppy (which is not an ordinary PNX file system floppy disc in that not all of it can be mounted), you will have to use the device */dev/flop* as follows:

1 Insert the disc in the drive and give the command:

cp /dev/flop */flop1*

where */flop1* can be any filename in some convenient part of the file system of the fixed disc. The time taken by *cp* to read the floppy disc depends on the interleave but for a boot floppy takes about 2.5 minutes. After reading from the floppy you will get the irrelevant message:

cp: read error

2 Type:

ls-ls */flop1*

You should see that the file is exactly 1001 blocks long. You can now replace the floppy with one that you wish to overwrite (it must have been formatted (see *fl*(1)), but you do not need to use *mkflop* on it since you are going to do a block for block copy)

3 Give the commands:

cp */flop1* **/dev/flop**
sync

There should be no messages, and approximately 2.5 minutes later, after the *sync*, you can remove the floppy disc and the file */flop1*

2.3.3.2 *File space*

The fixed disc quickly becomes full if you do not ask users to throw away their old files. The message:

no space on dev 0/0

appears when the file system on the fixed disc is full. The same message with **1/1** indicates that the currently mounted floppy is full. To find the amount of free disc space on a device use the command:

df

It assumes the fixed disc is to be looked at by default.
The command:

du

can be used to total the number of blocks used within a directory and since each user has a separate directory under */usr* the following command shows you how many blocks are taken up with users' files

du -s /usr/*<

To find the space left on a PNX file system floppy disc (which can be mounted or not) use:

df /dev/flop

fsck(1), which is run at every initial boot, also tells you how many free blocks are left on a device.

After running PNX for some time you may notice some performance degradation. This is due to the free block list becoming untidy. To tidy the free list, enter single user mode, log in as superuser and issue the command:

fsck -sx

strip(1) removes the symbol table and relocation bits ordinarily attached to the output of the assembler and loader. This is useful to save space once a program has been debugged.

2.3.3.3 *File system consistency checks*

Because of the dangers of file corruption, the file consistency checking program, *fsck*(1), is run at every initial boot.

The file system on the fixed disc may be corrupted if the system crashes or if you power off or reboot the system without flushing out buffers to the disc. Any floppy mounted at the time may be corrupted too. The system does not, in general, notice that a file system is corrupt so you could carry on using it unaware until some major catastrophe.

If the data in a file is corrupted this cannot be detected automatically. However, if the blocks describing how files are distributed on the disc are corrupted the program *fsck*(1) can often find the faults and sometimes correct them.

Since *fsck*(1) reads from and writes to the fixed disc or floppy directly it should not be used on a mounted disc. In practice this means you should not mount a floppy before using *fsck* on it, and you should reboot and log in as the superuser in *single user mode* before using *fsck* on the fixed disc.

Appendix 8 contains an article describing *fsck*(1) in detail.

Checking the fixed disc

You must be logged in in single user mode.

If you give the command:

fsck

the program looks in the file */etc/checklist* for the names of the discs it should check. This file normally contains the string */dev/hard*, so the command checks the fixed disc. When it finds an inconsistency or error it provides an error message and, usually, prompts with a suggestion as to what it should do to recover from the fault.

It is usually best to just reply **y** to all the prompts. It is difficult to do better than this program at patching up discs. However, you should take note of the filenames it mentions as you may find that these have had to be removed from the file system. Even old and rarely used files can be lost. Sometimes files lose track of which directory they should be in and *fsck*(1) may put them in the directory */lost+found* for you to look at and decide where they should be.

If you say **n** to the prompts from *fsck*(1) the fault remains in the file system. A *no* reply to all the prompts will mean that no changes will be made to the disc.

You can reply *yes* or *no* automatically to all the prompts by putting **-y** or **-n** after the fsck command. Some errors, such as losing some free blocks, are not particularly severe, but if you do suspect a fault you should always run *fsck*(1) as even a minor fault can rapidly cause disruption throughout the file system.

If *fsck*(1) makes any changes to the fixed disc it asks you to reboot immediately without issuing a *sync*; this is because the PNX system does not know about the changes that *fsck*(1) has made to the file system, and a *sync* causes it to overwrite some of them incorrectly.

If *fsck*(1) does not find any errors you are not prompted for **y** or **n**. *fsck* finishes with a summary of the system:

N files *m* blocks *x* free

where *N*, *m* and *n* are numbers showing, respectively, the number of files, number of used blocks and number of free blocks in the system.

Just type **^Z** to move to multiuser mode.

Checking floppy discs

To check a PNX file system floppy disc you do not need to go to single user mode, but it does not matter if you do. Make sure that the floppy is not mounted (the easiest way is to unmount it and ignore any error message). Give the command:

fsck /dev/flop

In this distribution *fsck*(1) is not permitted to ordinary users, but since they should be encouraged to check their own floppies with it you will probably want to change the execute permission on *fsck*

chmod a+x /etc/fsck

They will not be able to use it on the fixed disc because they do not have write permission to */dev/hard.*

Normally, PNX expands directories to hold file references as they are added. However, a directory is not contracted once references are removed. *fsck* cannot enlarge directories so not only must you create a lost and found directory (*/fd/lost+found* on a mounted floppy) you must also create several dummy files in the directory and then remove them to make sure there are empty slots for the homeless files that *fsck* finds.

2.3.3.4 *Other disc errors*

The *fsck*(1) program above cures errors in the file system caused principally by the software not managing to make several changes on the disc instantaneously. Considerably more dangerous are the errors caused by the hardware failing to write a proper block on the disc due, for example, to power failure.

Each block of data on the disc has a checksum kept with it which is used to give confidence that the data has not been corrupted when you try to read it back. If it has been corrupted you get one of two error messages:

Unrecoverable error at block *xxxx* **of the hard disk**

Error at block *xxxx* **of the hard disk, recovery after x retries**

where *xxxx* is a hexadecimal number. In the first case the *read* call returns a value of -1. In the second case the read is successful after x retries. For most status numbers, if this recurs repeatedly when trying to read a particular block you can only recover by rewriting that block with new data, as the original content is likely to be unknown (*dd*(1) can be used to write single blocks on the disc, if you know what should be there).

Apart from corruption of the contents of blocks, more severe errors can cause the headers of the blocks to become unreadable. This means that the whole track or cylinder containing that block must be reformatted. If the disc is a floppy this means reformatting the whole disc with *fl*(1). If the fixed disc needs reformatting, consult an ICL engineer.

To provide greater confidence in the correct working of the system, when an object file is produced a checksum of its contents is added to the file. If you get the error message:

xxx **- checksum fail - csum=1F5CF663, sum=1F585663**

where *xxx* is the name of a program, it means that an exact copy of the program was not loaded into memory. If the message recurs every time you use the same program you should recompile the object code or copy the utility from the PNX distribution floppies again.

2.3.4 Network management

PERQ can be part of a wide area network through its RS232C interfaces or part of a local area network through its Open Systems LAN interface. Setting up the physical connection is not covered here. For details consult one or all of the following:

1 The remote site manager

2 The local PTT authority

3 Modem manufacturers and suppliers

4 The ICL support representative

Chapter 7 lists the communications facilities available. Three of these facilities require superuser maintenance; *uucp*(1), ECMA-72 transport service procedures and *mftp*(1). *uucp*(1) requires four files set up with entries describing the remote systems. Programs that use the transport service and *mftp*(1) need to be listed in the transport address file.

2.3.4.1 *uucp*(1)

The file */usr/src/cmd/uucp/READ.ME* contains general information about *uucp*(1). The file */usr/src/cmd/uucp/PERQ_INFO* describes those aspects of the implementation specific to PNX.

The following files must be tailored to each individual system by the system manager:

/usr/include/whoami.h	host system name
/usr/lib/uucp/L-devices	device connection information
/usr/lib/uucp/L-dialcodes	dial information
/usr/lib/uucp/L.sys	access information to remote systems
/usr/lib/uucp/USERFILE	access information for remote systems and local users

These files are generated when the *uucp* system is *primed* from floppy disc and already contain the required entries for PERQ to PERQ communication.

The forms entries should take are described in the relevant articles on *uucp* in *UPM*(2B). Example entries are the general information files above.

In addition to these files, */etc/passwd* must contain an entry for *uucp* to allow a remote login from another system and to invoke the *uucp* program in the responding PERQ. The *PERQ_INFO* file describes how to do this.

2.3.4.2 *Transport address file*

The transport address file, also called the nameserver file, is a complete list of all the services running in remote machines on the Open Systems LAN that the PERQ can access, together with a list of the services the PERQ offers to other machines on the network.

An entry in the transport address file for a remote service may be considered to be an agent for that service. Services offered in the PERQ may only be accessed by agents in other machines.

Each entry in the transport address file has the following format:

service name, service TSEL, dummy, network address

The network address is a three part number, each part separated by single commas. Each part is an integer in the range 1 to 65535. Each machine on the network has a unique network address embedded in the network hardware. You need to know this address and the addresses of all other machines on the network to set up the transport address file.

The TSEL identifies the service address of a service within a machine. The number is an arbitrary integer within the range 1 to 65535 and acts like a phone number to that service. Each service within a machine must have a unique TSEL but services in remote machines may duplicate the TSEL in the local machine.

It is best to group all the agents together as entries in the transport address file. These entries have the form:

agentname, TSEL in remote machine, agent ID, network address of remote machine

You need to know the service offered by each remote machine and the TSEL of each service in each machine. Each PERQ on the network should offer a *slaveft*(1) service. To be able to run *mftp*(1) you need an agent in the transport address file for each PERQ that you want to connect to. The name of the agent is irrelevant but for convenience should identify the service required, for example, *PERQBFTP*. The agent ID is not relevant in this release. For convenience, always set it to 1.

The next set of entries should be the services offered by the local machine. The entry for a service has the form:

servicename, TSEL, dummy, network address of this machine

The name of the service is arbitrary but should indicate the service offered, for example, *PERQAFTP*.

Creating the transport address file

1 Log in as the superuser

2 Run *mknet*(1). *mknet* creates the transport address file and all the agents and services listed in the transport address file as special files in the directory */lan*

3 Respond y to the request to create a new transport address file.

If at this stage *mknet* fails do not try to run it again but run *fsck*(1) to find out what is wrong (see section 2.3.3.3).

mknet returns with the message:

new (empty) transport address file formatted

if it is successful at this stage

4 Enter replies to the following prompts for each agent and server you want to enter in the transport address file:

(a) **supply special filename** <RET> to terminate l to list

Type the filename for the agent or service followed by RETURN

(b) **TSEL-ID of service (which may be a different file)?**

For an agent, give the TSEL of the service in the remote machine, followed by **RETURN.**

For a service, give the TSEL of the service in this machine, followed by **RETURN**

(c) **Does the file offer a service - (reply Y or N)?**

Respond y if the file is a service in this machine, followed by **RETURN.**

Respond **n** if the file is an agent for a service in a remote machine, followed by **RETURN.**

The next two prompts occur if you reply n

(d) **TSEL-ID to identify caller when requesting service?**

This value is not significant in this release so just give a response of 1 followed by **RETURN**

(e) **Address of M/C providing service**

Respond with the three part network address of the remote machine offering the service (you may respond with one integer if the first two are 540, 31744)

5 When you have entered all the files just type **l** in reponse to the filename prompt.

This instruction lists out the transport address file so you can check that you have entered everything.

The filename prompt is repeated so you can add entries if you wish

6 To terminate, just type **RETURN** in response to the filename prompt

Filenames established in the transport address file are also entered as special files in */lan.* The files corresponding to agents for the FTP program may be used as parameters to *mftp*(1). If the PERQ offers an FTP service, both POS PERQs and other PNX PERQs may call the machine. POS PERQs use PERQFTP (see *ICL PERQ: System Software Reference* (R10101, Edition 4). For details on using the ECMA-72 transport service see *ICL PERQ: Establishing IPA under PNX.*

Updating the transport address file

1 Log in as the superuser

2 Run *mknet*(1).

If a transport address file already exists, *mknet*(1) first checks that all the entries in */lan* are also entries in the transport address file.

This consistency check is to make sure that there are no illegal entries in */lan* created using *mknod*(1).

For each special file in */lan* for which there is no entry in the transport address file *mknet* gives the following series of prompts:

(a) **no entry for special filenames <names>, c to create entry, d to delete.**

Type **d** followed by **RETURN** to delete special file.

Type **c** followed by **RETURN** to create the special file as an entry in the transport address file.

Creating the entry leads to the following prompt

(b) **TSEL-ID of service (which may be a different file)?**

For an agent enter the TSEL of the service in the remote machine, followed by **RETURN.**

For a service enter the TSEL of the service in this machine, followed by **RETURN**

(c) **does the file offer a service - (reply Y or N)?**

Respond **y** followed by **RETURN** if the file is a local service.

Respond **n** followed by **RETURN** if the file is an agent for a remote service.

The following prompts only apply to agents

(d) **TSEL-ID to identify caller when requesting service?**

This number is not significant in this implementation so just respond **1** followed by RETURN

(e) **address of M/C providing service.**

Respond with the three part network address of the remote machine

After checking entries in */lan* against the transport address file, *mknet*(1) then checks entries in the transport address file against */lan* and deletes any spurious entries

3 Once the consistency check is complete, *mknet* prompts you for new entries just as in creating the transport address file

(a) **supply special filename <RET> to terminate l to list.**

Enter the filename followed by **RETURN** of the service or agent you want to create or change.

If the filename is a new entry *mknet* first checks that there is space in the transport address file

(b) For new files *mknet* prompts:

new entry, c to create entry, otherwise no change

Type **c** followed by **RETURN** to create an entry.

mknet then supplies the prompts for creating an entry given above. Any other response or any failure of *mknet* to create the entry returns you to the filename prompt

(c) For existing files *mknet* displays the existing entries and prompts for changes.

For a service *mknet* displays:

TSEL-ID of service is <n>
file offers a service
d to delete, c or u to update - otherwise no change

For an agent *mknet* displays:

TSEL-ID of service is <n>
file uses TSEL-ID <n>
to call machine address <n>
d to delete, c or u to update - otherwise no change

Respond **d** followed by **RETURN** to delete the whole entry.

Respond **c** or **u** followed by **RETURN** to change the entry, for example, it may be necessary to change the network address of a remote machine if that machine has had a new I/O board fitted.

mknet prompts for updates in the same way it prompts for new entries

4 Terminate the update by pressing **RETURN** in response to the filename prompt

2.3.5 Device management

Chapter 6 describes the way in which devices are added to the PNX system and Table 6.1 lists all the devices that PNX knows about. Some of the devices are already declared to PNX but others have alternative declarations depending on the way you want to set up the system. You must not create any special files with the major and minor device numbers listed in Table 6.1 unless they are the devices suggested by Table 6.1. Note that all major and minor device numbers have been changed from previous releases and you should change any user created special devices accordingly. See Chapter 6 for full details.

The PERQ has only one display, keyboard and tablet, yet under PNX it could be running several linked or independent processes at once, each of which may need to use all the PERQ components. The display and input components must, therefore, be shared between each process.

The PERQ high resolution display is ideally suited to multiple concurrent use since it can be partitioned into independent screen areas, known as *windows,* each window being used by a separate process.

Each of the input devices, the keyboard and the tablet, can only generate one input at a time so these components must be shared sequentially. You select which process is to receive their inputs at any one time.

The sharing of these PERQ components is controlled by the PNX Window Management System (hereafter referred to as WMS). The WMS provides two distinct functional interfaces: a procedural interface allowing programs to create and use windows as though they were standard UNIX terminals and a procedural interface to control the screen layout and the sharing of input devices. A supplied standard user interface program known as Window Manager uses this interface to allow you to create and control windows.

The WMS provides functions to support graphic operations within windows, and interfaces are available to permit microcode entry to windows for graphics output.

A further facility is a pop up menu package that allows a user process to display a menu of commands in a screen window. You can then use the puck to select one of the displayed commands. The WMS user interface uses this pop up menu package to provide the Window Management menu.

The WMS identifies the special device */dev/tty* with the window running a particular process. Additional special devices are provided; */dev/window* to allow direct access to a particular window and */dev/screen* to allow direct access for the graphics to the screen. */dev/screen* can be used for text as it uses the */dev/console* driver but results may be unusual. It is strongly recommended that you use the WMS for all graphic output and not */dev/screen* directly. In this release, updating on screen images is quite efficient and there is little performance gain in using */dev/screen.* Both */dev/window* and */dev/screen* are described in detail in Chapter 6.

When the WMS is not running the entire screen may be used as a terminal and */dev/console* and */dev/tty* refer to the whole screen. The special screen device */dev/screen* may be used for graphics.

Error messages from the kernel cannot be output to the screen while the WMS is running so PNX maintains another special file */dev/error* for kernel error output (see Chapter 6).

3.1 Windows

When able to receive input from the keyboard and tablet, a window becomes a virtual device which may be used as if it is a standard UNIX terminal, or as a PERQ screen supporting graphic operations or as a combination of the two. All windows are totally independent and receive output independently and concurrently. Programs using windows are only aware of their own windows.

Many windows (up to 31, depending on memory) may be visible at once. Windows may overlap, in which case one or more windows may be obscured in the area of overlap. Windows may be partly off the display. The Window Management System maintains the contents of windows off the display or obscured by other windows so that they can be reconstituted as soon as you make them visible again.

Through the standard user interface program (Window Manager) you determine which window is to receive input from the keyboard and tablet and control the display through moving windows, creating or destroying windows, changing their size or determining which of a series of overlapping windows should be visible.

3.1.1 Window properties

Each created window has a set of initial properties to define that particular window. The set of properties defining a window can be assigned using the standard user interface program, the Window Manager, or can be assigned from a *window description.*

A window description is a specially structured file containing information that can be used to establish a window. It is comparable with a PNX special file and can be made into an active window using the *open*(2) system call. Any number of window descriptions may be established in a file store, at any appropriate points in the file hierarchy.

There are two methods of establishing a window description:

1 By user program. The program creates a file in a specified format (see *wdesc*(5)) then issues a system call to *mknod*(2) to turn the file into a special file

2 By the *mkwind*(1) utility (see *mkwind*(1))

Windows are created with the specified initial properties whenever a program executes an *open* system call giving the name of the window description file.

Window properties fall into three categories; intrinsic, initial and dynamic.

Intrinsic properties, mainly derived from the hardware, are always true. The initial properties define the window when it is first created. Some of the initial properties may only be set from a window description. Some of these initial properties are also dynamic properties, that is, they can be changed by you or by a process after they have been set.

Intrinsic properties

The intrinsic properties of a window are:

1 The window pixels are one-to-one with the display pixels

2 The window origin has coordinates of (0,0) and is at the top left of the usable area. This remains true even if the window is moved, that is, the coordinates are always relative to the window, not the display

The tablet maps onto the whole of the display so that tablet coordinates map onto pixel coordinates on the screen. This is necessary so that the window manager can determine the window at which you are currently pointing. Tablet coordinates are given to the program (by the system call *wgread*(2)) relative to the origin of the window at which you are pointing, and relative to the whole display, and as absolute tablet coordinates.

Initial properties

The initial properties of a window are:

1 SIZE The size of a window is the height (in pixels) and the width (in pixels) of the area available for input or output, that is, the usable area. This does not include the title or the border area

2 POSITION The display coordinates required for the window's origin, that is, the top left corner of the usable area

3 CURSOR The hardware cursor pattern to be used when the tablet input is directed to the selected window

4 CURSOR FUNCTION This specifies the background colour of the window and the way in which the cursor pattern combines with the background pattern (see *window*(4))

5 CURSOR MODE This specifies the relationship between puck movements on the tablet and cursor movements on the screen (see *window*(4))

6 TITLE Windows may be titled or untitled. If a title is requested, it is placed above the usable area in a strip of colour the inverse of the window. The height is equal to the window font with a blank scan line above and below. The text of the title is aligned to the left text margin in the window

7 BORDER If a border is requested, the usable area, and title if present, is surrounded by a border several pixels wide in colours to ensure:

(a) a contrast with both the background window colour and the background screen colour

(b) the potential for emphasis to distinguish the selected state from the unselected state

8 RANK Where windows overlap, the window with the lowest *rank* is visible, and it obscures the other(s). The initial rank given in the window description is interpreted as relative to the lowest current rank in use. Rank increases away from the viewer; the lowest rank is zero

9 FONT The font to be used for standard text output (though text can also be output as graphics by use of *wdbyte*(2))

10 GENERIC/UNIQUE If a window is unique, that is, shareable, only one instance of that window exists. If more than one process is using the window, output is interleaved. If a window is generic, every call to open that window creates another instance of the window. However, even generic windows can be shared using *dup*(2) and *fork*(2). Set by window description only

11 DELETE/PRESERVE When a window has been closed by all the processes sharing it, it may be automatically deleted from the screen or preserved until deleted using the user interface. All windows created using the interface are automatically deleted when closed. Set by window description only

Terminal control parameters (see *tty*(4) and *ioctl*(2)) are initialised to default values, except for line width and page size, which are automatically initialised to match the window size.

Dynamic properties

While a window is in use, a process can alter:

1 Terminal control parameters, using *ioctl*(2)

2 The hardware cursor pattern, function and mode, using *ioctl*(2), (see *window*(4))

3 The mode for updating the window display (see *window*(4))

4 The window input mode

and you can alter the window's:

1 Position

2 Rank

3 Size

using the Window Manager.

3.2 Standard user interface (Window Manager)

The interface program enables you to create windows of any size in any position, move windows, change the rank of a window to make it visible and select and deselect windows for input from the keyboard and tablet. You can terminate the WMS by deleting the system window.

You control the system by inputting commands to the system window from the keyboard or selecting commands from a pop up menu using the puck. You may also control the display using the puck.

3.2.1 Entering the Window Management System

Start the WMS by issuing *winit*(1). On initialisation, the WMS window appears in the top right-hand corner of the screen. *winit*(1) first displays the system window then reads the *wprofile*(5) file to initialise WMS to the required configuration. *wprofile* determines the background colour of the screen, the logical to physical button mapping and which windows should be opened and displayed immediately.

The default WMS configuration is:

a white screen

logical button A = physical button 0 (left on the three button puck, yellow on the four button puck)

logical button B = physical button 1 (middle on the three button puck, white on the four button puck)

logical button C = physical button 2 (right on the three button puck, blue on the four button puck)

WM Escape = ^**RETURN**

Note: The default configuration activates a fourth button with identical effects to the third button. If using the optional tablet you need to change *wprofile*(5) if you want the fourth button to have its own effects.

At the bottom of the WMS display is the *echo line*. This is a small area of the display the full width of the screen of height equal to the system font. All text typed at the keyboard is echoed to this area. When you press **RETURN** the text is input to the selected window as it is read by the program running in the selected window. Every window has a text cursor, an inverted square to show where the typed input will appear.

Windows are superimposed on a shaded background area. If the screen colour is inverted the background is a dark grey, if normal the background is a light grey. Inverting the screen colour inverts the window colour specified in the cursor function property but the contrasts between open windows, closed windows and the background area are always clear.

Figure 3.1 shows a typical window management display with the system window foremost and selected for input.

3.2.2 Getting help

Initially the system window displays the message:

type h or H for help

Issuing this command results in a help window displaying general information describing the effect of puck button presses, the way to enter commands and the way to select a window to receive input. This display also tells you how to get a pop up menu of available commands.

The pop up menu includes a HELP command. Selecting this command results in another help window listing all the available commands with instructions and a description of their effects.

The HELP key sends a signal to the process running in the selected window so the process can decide what to do about help.

3.2.3 Selecting a window

There are two devices for input; the keyboard and the tablet and puck. A window can only accept input from a device if it is first *selected* to receive input from that device.

When you first enter the WMS the system window is already selected, that is, ready to receive input. The selected window is always distinguishable from other windows by an emphasised border.

The keyboard is always attached to the currently selected window. Anything you type is initially echoed to the echo line, but after you press **RETURN**, the typed input appears in the selected window as it is read by the program.

The tablet, however, may or may not be directed to the currently selected window. Initially the tablet is *uncommitted*, that is, input from the tablet is directed to the interface program not the selected window. The cursor pattern indicates whether the tablet input is uncommitted or directed to the selected window.

The default cursor pattern for the uncommitted state is shown below.

 (actual size)

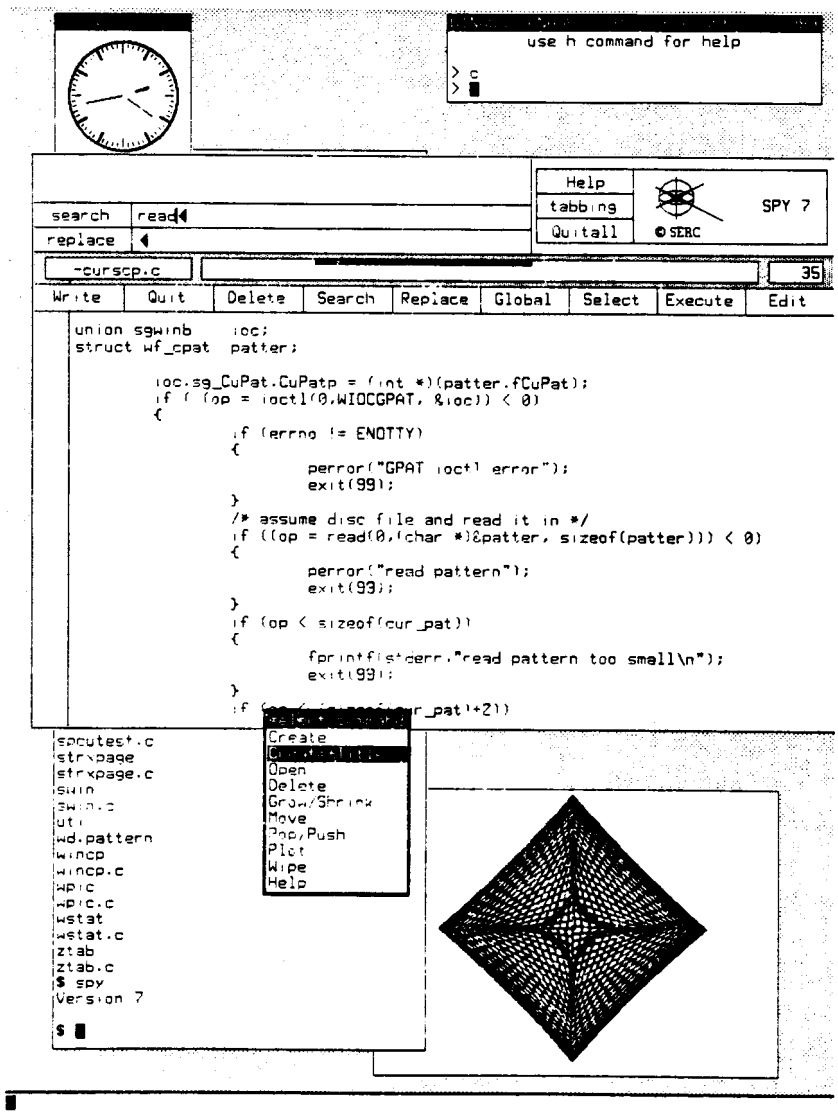


Figure 3.1 A typical Window Manager display

When the tablet is in this uncommitted state, you may select a window by moving the cursor to any visible part of the required window and then briefly pressing and releasing button A (left or yellow) on the puck. When the button is pressed the cursor changes to the default selected state. When released, the window pointed to is selected and the border of the selected window is then emphasised. There are two possibilites:

1 If the input mode is IMNOINT (*window*(4)), the process using this window has no interest in tablet input so only keyboard input is connected to the window. The cursor reverts to the uncommitted state

2 If the window input mode is not IMNOINT, both keyboard and graphic input are connected to the window. The cursor becomes the pattern specified in the window description or the default selected cursor remains. The default selected cursor is shown below

 (actual size)

The system window may be selected as any other window. The system window can be both selected and brought to the foreground by pressing *Window Manager Escape* while in the uncommitted state. The default Window Manager Escape key is ^**RETURN**. This key is configurable and may be set in *wprofile*(5).

3.2.4 Deselecting a window

As long as the tablet is in the uncommitted state you may deselect a window by simply selecting another window.

If the tablet input is selected, pressing *Window Manager Escape* always causes a change to the uncommitted state. It is also possible to enter the uncommitted state by moving the cursor to any part of the display which is not part of the selected window, provided that the cursor mode is CMAUTO. Note that while the tablet input is selected, the keyboard input is also selected (to the same window), but that the keyboard input remains attached to the selected window even if the tablet becomes uncommitted.

The WM menu appears in its own window already selected for input. If you do not want to select a command you can get rid of the menu by selecting any other area of the screen. The resulting selected window is the window originally selected before the menu appeared.

3.2.5 User interface commands

There are three ways to issue commands to the Window Management System:

1 Select the system window and type commands followed by **RETURN** at the keyboard

2 Press and release button A (left or yellow) twice in quick succession or point the cursor at the background area and press and release button A (left or yellow) once to obtain a pop up menu of available commands. While the pop up menu is available the tablet input is selected. Point the cursor at the required command and press and release button A (left or yellow) to select it.

Selecting anywhere outside the menu causes the menu to disappear with no command selected

3 Some commands may also be implemented by puck button presses when the tablet is uncommitted

Table 3.1 summarises the commands available, how they are effected and their effect:

Table 3.1
Interface commands

Command	*Effect*	*Effected by:*	
CREATE	Creates a window	1	Typing **C** or **c** at the keyboard
		2	Selecting the **create** or **create+title** command from the menu
OPEN	Opens a window description file causing the window to be displayed	1	Typing **O** or o at the keyboard
		2	Selecting the **open** command from the menu
DELETE	Deletes a window from the screen	1	Typing **D** or **d** at the keyboard
		2	Selecting the **delete** command from the menu
GROW/SHRINK	Changes the size of a window	1	Typing **G** or **g** or **S** or **s** at the keyboard. The four commands are equivalent
		2	Selecting the **grow/shrink** command from the menu
POP/PUSH	Changes the rank of a window to bring it to the foreground or move it to the background	1	Selecting the **pop/push** command from the menu
		2	Pressing and releasing button B (middle or white)
MOVE	Moves a window to a part of the PERQ screen	1	Selecting the **move** command from the menu
		2	Pressing and releasing button C (right or blue or green)
PLOT	Plots a window to a file on the PLOT spooler ready for output to a printer	1	Typing **P** or **p** at the keyboard
		2	Selecting the **plot** command from the menu
WIPE	Deletes all closed windows from the screen	1	Typing **W** or **w** at the keyboard
		2	Selecting the **wipe** command from the menu

3.2.5.1 *Creating a window*

1 Give the required command either by selecting the system window and typing the command followed by **RETURN**, or by selecting from the pop up menu. If selecting the command, select the **create** command if you do not want a title or the **create+title** command if you do. The menu changes to show a prompt asking you to type the title and the cursor changes to show that the program is requesting keyboard input:

 actual size

Just type the title followed by **RETURN**. The title will be all the letters typed up to **RETURN**.

If typing the command, the syntax is one of:

c *title*
C *title*

where *title* is the name of the window you want to create. *Title* must be one word with no spaces. The maximum length for a window title is 70 characters. Titles longer than the width of the window are truncated to fit the specified width and are displayed aligned with the left text margin. The lower case command creates the window whereas the upper case command both creates and selects the window.

2 Once you have selected or typed the command the cursor changes to the create cursor:

(actual size)

Position this cursor at one corner of your intended window. Press and hold down button A (left). The cursor changes to the special box cursor. As you move the cursor, lines are drawn between the fixed cursor and the current cursor position to show the resulting user area of the window. Releasing the button fixes the box size and creates the window.

A shell is initiated to use the window as a controlling terminal. You are already logged in, and the directory is that under which you were logged in when the WMS was started. *login*(1) and *cd*(1) are of course available to change the directory in use by that shell.

That shell has the following standard files open:

Description	*File*
0	Input file for reading messages typed when the associated window is selected
1	Output file to that window
2	Error output file to that window

Each new shell executes a **.profile** file if it exists and environment parameters exported from the shell before *winit* was invoked appear in the environment of the window shells.

3.2.5.2 *Opening a window*

Give the required command either by selecting the system window and typing the command followed by **RETURN**, or by selecting the command from the menu.

If you select the command the menu changes to request the filename of the window description and the cursor changes to show that the program is requesting keyboard input. Just type the filename followed by **RETURN.**

If you type the command, the syntax is one of:

o *filename*
O *filename*

where *filename* is the name of a window description file created using *mkwind*(1). The lower case command opens the file and displays the window. The upper case command opens the file and both displays and selects the window.

A window is created to the specification given in the file whose name is *filename*. A shell is initiated in that window as for the create command (see above).

3.2.5.3 *Deleting a window*

Give the required command either by selecting the system window and typing the command followed by **RETURN**, or by selecting from the pop up menu. A distinctive cursor pattern appears (see below). Move it to a visible part of the window to be deleted, and briefly press and release button A (left or yellow).

The **delete** command always deletes the window but the processes running in that window are not necessarily terminated. If the window is a control terminal, **delete** sends a hangup signal (SIGHUP) to all the processes using the window. For a non-control terminal, **delete** sends the SIGWSTS signal. Processes may or may not be terminated by these signals. To a process, output to a deleted terminal window and *ioctl*(2) calls always appear to be successful but input produces **EOF**. However, graphic output to a deleted window always fails unless only user store areas are involved.

When all the processes using the window are finished, the window is closed. Once closed, the window is either automatically deleted, if the delete property is set, or remains on the screen so that the contents are visible but the background is shaded to differentiate the closed window from active windows. You can delete a closed window in the same way as for ordinary windows. Alternatively select the **wipe** command from the pop up menu. This command deletes all closed windows.

3.2.5.4 *Changing the rank of a window*

Position the uncommitted cursor in the window you want to pop/push and press and release button B (middle or white). The window pointed at is then brought to the foreground or pushed to the back if it was already in the foreground.

Alternatively, select the **pop/push** command from the pop up menu. The cursor changes to the pop/push cursor:

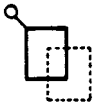
(actual size)

To bring a window to the foreground, position the cursor in any visible part of the window and press and release button A (left or yellow). All other windows remain in their relative positions. If the window pointed to is already in the foreground it is pushed to the background, that is, given the highest rank, and all other windows have their rank reduced by one.

3.2.5.5 *Moving a window*

To move a window with the puck, first ensure the tablet is in the uncommitted state. Move the cursor to a visible part of the window. Press and hold down button C (right or blue or green); the cursor becomes a drawn box, the size of the window and tracks the puck. Move the puck to move the window shaped cursor to the required position. Fix the new position of the window by releasing button C (right). The actual window is moved to the position shown by the cursor.

Select the **move** command from the pop up menu. The cursor changes to the Move cursor:



(actual size)

Move this cursor into the window you want to move. Press and hold down button A (left). This fixes the initial point of the move. Move the cursor in the direction you want to move the window. The cursor becomes a drawn box, the size of the window to demonstrate the potential effect of the move. Releasing the button fixes the final position of the window.

Moving a window does not change its rank. It may therefore become partly or totally obscured. Also, the window may be moved partly off the display; it cannot be moved totally off the display, so there is always a part of it available in order to move it further back onto the display.

After moving a window tablet input becomes uncommitted or becomes recommitted to the window selected before the menu was requested.

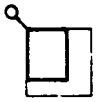
3.2.5.6 *Changing the size of a window*

Give the required command by selecting the system window and typing G or g or S or s at the keyboard or by selecting the **grow/shrink** command from the pop up menu.

A window may be thought of as having nine regions as follows:

Top left	Top Side	Top right
Left side	Centre	Right side
Bottom left	Bottom side	Bottom right

The area pointed to by the size cursor (see below) determines the type of change of size:



(actual size)

If the cursor is placed in a *corner* area, pressing button A (left or yellow) changes the cursor to a dynamic corner of a drawn box with the fixed corner being the opposite corner of the window. Releasing button A (left or yellow) fixes the new corner.

If the cursor is placed in a *side* area, pressing button A (left or yellow) makes that side dynamic. Moving the cursor moves the side. Releasing button A (left or yellow) fixes the new side.

If the cursor is placed in the *centre* area, pressing button A (left or yellow) makes the bottom right corner dynamic. Releasing button A (left or yellow) fixes the corner.

The window can grow so that a large amount is off the screen (windows may be larger than the screen).

A SIGWSTS signal is sent to all processes using the window. The contents of the window are redrawn to realign text and graphics with the new origin.

3.2.5.7 *Plotting a window*

Give the required command by selecting the system window and typing **p** or **P** at the keyboard or by selecting the **plot** command in the pop up menu.

The PLOT cursor appears:

(actual size)

Move the cursor to any visible part of the window to be plotted and press and release button A (left).

The command activates the program *splot* which plots the window contents to the plot spooler for output to a printer. The printer must be on line.

3.2.6 Configurable options

Wprofile(5) is an editable text file controlling certain WMS features:

1 Background screen colour. If the screen is white, normal windows are white, inverted windows are black. If the screen is black, normal windows are black, inverted windows are white

2 Logical to physical button mapping. Defaults are:

Logical A = physical 0 = left = yellow
Logical B = physical 1 = middle = white
Logical C = physical 2 or physical 3 = right = blue or green

Logical D is not generated in the default setting. You may need to make physical button 2 set logical D in addition to logical C to provide backward compatibility for programs written for a four button puck now running with a three button puck but this may lead to unexpected results. You may need to configure the buttons to make logical D = green on the four button puck

3 Window Manager Escape key. Default is ^RETURN

The way in which the tablet is declared under */etc/mknod* determines relative or absolute tracking (see *tablet*(4)).

Cursors used by WMS are listed in *wcurs*(5). These patterns may be changed using *cedra*(1). The cursor pattern may also be changed by programs using *ioctl*(2) calls (see *window*(4)).

Natural language text in menus, error messages and help is initialised from filestore files so that it may be changed.

The files are:

/etc/wdev/SEVHELP
/etc/wdev/comhelp
/etc/wdev/commenu
/etc/wdev/comerror

3.3 Software interface

A process uses the WMS by issuing system calls.

A process can only receive input from a window or output to a window that is *active*.

A process makes a window active by use of the *open*(2) system call, specifying the name of a window description special file.

If the window type is *shared* (unique) then the window may already be active. As output to a shared window is interleaved, input is on a "first come" basis, and control actions are on a "last setting" basis, it follows that unplanned sharing of a window may produce unexpected results.

Normally, a window is generic and each *open*(2) call to a window description file causes an example of the window to be displayed on the screen. The window is not necessarily visible, however.

The *close*(2) system call deactivates a window making it unavailable to the closing process. A window is closed when all the processes using it have issued the *close*(2) call. A closed window may or may not be deleted from the screen. If not it is shaded a darker grey than the background to mark it as closed. This allows you to inspect the contents at leisure. A closed window can only be reopened using */dev/tty* or */dev/windowN*.

PNX provides a number of system calls that processes may use when controlling a window. These system calls fall into two categories:

1 TEXT If a window is to be used for the output of standard text only, the procedural interface is identical to that for a standard UNIX terminal

2 GRAPHICS Additional procedures are available to enable programs to:

(a) Manipulate pixel patterns, both on the display and also in main memory (for example, to prepare patterns for later display)

(b) Use the graphics input device (the tablet and puck)

(c) Present a pop up menu

A process can also gain direct access to a window using the special device */dev/window* (see *window*(4) and Chapter 6).

All the system calls, and so on, mentioned in the following sections are defined either in the *UPM*(1), or in Appendix 3 of this *Guide to PNX* if they have been modified or added for PNX.

3.3.1 Text interface

A process can output to a window with the *write*(2) system call, and can receive input from the keyboard with the *read*(2) system call when you permit it by selecting that window (see section 3.1.3.1). The process can also control certain aspects of the window with the *ioctl*(2) system call, as if the window were a standard UNIX terminal (see *tty*(4)).

The way in which a process uses a window for text is almost exactly identical to the way in which it would use a standard UNIX terminal. Thus, for example, most C programs which output text only, can be transferred in source form, without alteration, to PNX.

A limited number of intelligent VDU functions have been implemented. The following character combinations have special effects when written to a window (or to the screen as a console):

ESC X <32 + n>	Go to column n
ESC Y <32 + y>	Go to line y
ESC K	Clear the screen and go to column 0, line 0
ESC <other>	Ignored

Line and column spacing are those of the standard font.

3.3.1.1 *Keyboard input*

Keyboard input is always directed to the currently selected window. WMS interacts with PNX to produce the expected results from control key input and is interpreted as described in *tty*(4).

When data is being output to a window, rack-up stops when the window is full, to give you time to see all the output. Press:

1 **RETURN** to permit the next line to appear

2 **SPACE BAR** to permit the next page to appear

3 **^P** to allow output to continue without pausing until interrupted by an input action

3.3.2 Graphics interface

- A process can send graphical output to a window in the following ways:

1 STRAIGHT LINE A straight line can be drawn between any two points in the usable part of the window by using the *wline*(2) system call

2 CHARACTERS The *wdbyte*(2) system call can be used to write characters in a chosen font from a supplied string starting at a specified point in the window. The process continues until one of the following occurs:

(a) The string is exhausted

(b) A right-hand boundary is encountered

(c) A control character is encountered in the string

The parameters are updated and control is returned to the user program, which should then determine the reason for the termination, and take any further action required

3 PATTERNS The *wrasop*(2) system call can be used to copy a rectangle pattern from elsewhere in that window or in main store, or may be used to logically combine such a pattern with the existing contents of the rectangle, depending on the parameters supplied to *wrasop*(2)

wline, *wdbyte* and *wrasop* may be used to write to a user process store area. Transfer of data to and from a window is clipped to the window boundaries and may be clipped further by user specified parameters.

See *rectangle*(3) for instructions on obtaining store at run time for use in conjunction with these three system calls.

The process can also control certain aspects of the window, with the *ioctl*(2) system call, in a way relevant to graphical use of the window (see *window*(4)).

User clipping

Sometimes you may wish to specify graphic output for a wide area while limiting the visible affects to selected parts of a window. This is assisted by *user-clipping parameters* to the system calls *wline*(2), *wdbyte*(2), and *wrasop*(2). These user clipping parameters ensure that only the area of the window designated by those parameters is acted upon by the system call.

Performance considerations with wdbyte(2), wline(2) and wrasop(2)

As mentioned earlier, the window management system maintains an off-screen copy of each window, so that it can refresh the on-screen copy if that were to be obscured and then brought to the foreground. If a window is already in the foreground when one of the *graphic system calls* (*wdbyte*(2), *wline*(2) or *wrasop*(2)) is issued, window manager updates the image on screen. The offscreen copy gets refreshed later.

On screen refreshing can be suppressed and restarted by use of *ioctl*(2) (see WIOCNOUPD and WIOCDOUPD in *window*(4)). If a process needs to perform many small amendments invisibly, it should first use WIOCNOUPD, perform the amendments, and then use WIOCDOUPD to produce the amended display.

3.3.2.1 *Pop up menu package*

PNX provides a general purpose pop up menu package which enables a user process to display a menu of commands in a screen window. Each menu is displayed within a window surrounded by a single pixel border. If a title is supplied it is displayed at the top of the menu, its height equal to the font height. You can then use the tablet and puck to select the required command.

The system call *pickmenu*(2) displays the menu. Parameters to the system call specify the window to contain the menu, the position of the menu in the window, the commands to be listed, the font and a flag to determine whether the area of window used by the menu is to be saved in an off screen memory area and restored when the menu disappears.

Commands and their command numbers together with an optional title may be passed as an array of strings and numbers. Alternatively, you can set up a file containing the commands, command numbers and optional title (see *menufile*(5)) and use *buildtext*(3) to convert the file into an array which can then be used as a parameter to *pickmenu*(2). The following is an example menufile for the Window Manager menu:

Select command

0 Create
1 Create + Title
4 Open
6 Delete
7 Grow/shrink
9 Move
10 Pop/push
8 Plot
12 Wipe
11 Help

The position of the menu in the window may be determined in three ways:

1 The menu may be completely fitted to the usable area of the window

2 The menu may be placed at a position defined by the user

3 The menu may be placed at a specified position with a specified size as defined by additional parameters

Once a menu is on the screen the tablet input is selected. To select a command point the cursor at the required command and press down button A (left or yellow). This causes the command line to invert. Releasing the button selects the command so you may reposition the cursor to select another command while the button is still held down.

If you do not select a command you can dispose of the menu by:

1 Selecting an area outside the menu

2 Typing **Q** followed by **RETURN** at the keyboard

The menu may also be disposed of by the calling process using the routine *menuesc*(3).

The following example program calls menu routines with parameters supplied by the operator. Default parameters are set up initially and results are displayed.

Example

```
#include <stdio.h>
#include <wuser.h>
#include <sgwin.h>
#include <mencntrl.h>

#define  LS      1
#define  PWD     2
#define  PS      3
#define  STTY    4
#define  SWIN    5
#define  QUIT    99

char *shellcom[] = {NULL, "ls", "pwd", "ps-a/unixn", "stty","swin" };

unsigned selected;
unsigned entry_nrs[] = {LS, PWD, PS, STTY, SWIN, QUIT };

char *entry_text[] = {"ls", "pwd", "ps", "stty", "swin", "quit", NULL };

struct menu_ctl test_menu =     {
                                1,              /* standard output */
                                {"selection:",
                                  entry_nrs,
                                  entry_text,
                                };
                                0,              /* standard font */
                                &selected,
                                _FRMFIT
                                _SAVE
                                };

struct clipctl test_frame =     {100,
                                  200,
                                  100,
                                  200,
                                };

main()
{
 int res;
 while (selected != QUIT)
     {
      res = pickmenu (&test_menu, &test_frame);
      if (res < 0)
            {printf ("pickmenu error %d\n", res);
            exit(1);
            }
      if (res)
            system (shellcom[selected]);
      else
           printf ("nothing selected\n");
     }
}
```

3.3.2.2 *Tablet input*

Input modes

When you select a window to receive input there are two possibilities. If the input mode is IMNOINT only the keyboard becomes attached to the window and the tablet remains uncommitted. If the input mode is not IMNOINT then the tablet input is also committed. Once a window is selected, changing the input mode does not cause the tablet to become committed. You have to select the window again for the tablet input to be selected.

If a process makes a call to *wgread*(2) while the input mode is IMNOINT then the mode automatically changes to IMSAMPLE. In sample mode, a record is generated on every call of *wgread*, whether or not the window is selected, so any process can receive information on puck position and button presses. *wgread* also returns a flag indicating whether or not the window is selected so programs should check the flag to see if the input is meant for them.

Input records are generated whenever one of the following input modes is enabled:

1 SAMPLE (IMSAMPLE) One input record is generated immediately on each call of *wgread*(2) even if the window is not selected. This means that a program can receive input being directed to another window. This may be a problem for old programs that expect *wgread* to only return input if the window is selected

2 REQUEST (IMREQUEST) One input record is queued for the window on the first occurrence of a specified *trigger condition* (see below) after each call of *wgread*(2)

3 EVENT (IMEVENT) Input records are queued continuously for the window on every occurrence of a specified trigger condition after the mode is set

4 ASEVENT (IMASEVENT) As for EVENT but a signal is also generated each time an input record is generated. This allows a user process to be interrupted asynchronously by the occurrence of each trigger event

Trigger conditions are

1 Any puck button change of state

2 Any change of tablet coordinates by more than a specified amount

3 Elapsed time in units of 1/60th of a second

4 Availability of keyboard input for the window

5 Selection or deselection of the window by the operator

Hardware cursor modes

A user program can change the cursor pattern and function at any time. The function determines how the pattern combines with the underlying image (see *window*(4)).

The hardware cursor is really an output device, but its use as a tablet echo device makes it essential to describe its modes in conjunction with tablet input. Note that these cursor modes are only effective if the window input mode is not IMNOINT.

The *hardware cursor modes* available while a window is selected are:

1 AUTOTRACK (CMAUTO) The cursor tracks the position of the puck or pen, and the whole tablet is mapped onto the whole display. Window Manager monitors the coordinates. The process with access to the selected window only receives tablet coordinates within the window area, converted so that they are relative to the window's origin. The coordinates correspond to the displacement in pixels from the origin.

   If the coordinates go outside the selected window, the tablet immediately enters the uncommitted state

2 EXTENDED AUTOTRACK (CMXAUTO) The cursor tracks the position of the puck or pen, and the whole tablet is mapped onto the whole display, but the window manager does not monitor the coordinates. The process can receive coordinates from the whole of the tablet area. To enter the uncommitted state, you must press the Window Manager Escape key unless the process returns the cursor to AUTOTRACK mode

3 ON (CMON) The cursor appears at a position set by the process with access to the selected window; otherwise as for EXTENDED AUTOTRACK above

4 OFF (CMOFF) As for ON above, except that the cursor picture does not appear

Use of AUTOTRACK makes it easier to control a multi-window display with the window manager facilities, but ON is necessary when you want your process to control the cursor position on the screen by itself (for example, to direct you by pointing to various items). OFF is useful when you require the whole tablet but would find the cursor picture visually intrusive. CMXAUTO is useful when you want to use the whole tablet, for example, to trace a diagram.

Spy is a screen based, interactive, modeless text editor that is easy to learn and quick to use yet is an extremely powerful editing tool.

Screen based means that not only is the text you are editing available on the screen but also all the editing facilities are provided as a display. *Spy* is a point and act type editor. Moving the puck on the tablet moves a corresponding cursor on the screen. You position the cursor to point, either at a command or action you want or at the text you want to change and then effect the action by either pressing a puck button or just typing text wherever you want it to appear. Having all the facilities available on the display makes *spy* easy to learn as you do not have to remember a set of command names. *Spy* not only displays the available facilities but also maintains an area of the display to remind you of the current effect of pressing any one of the three puckbuttons.

Interactive means that all the editing actions you choose to make are immediately effected on the text in front of you. In other words, what you see is what you get. *Spy* not only gives you immediate feedback on your editing actions but also maintains areas of the display to let you know where you are in a file.

Modeless means that all the editing facilities are always available. *Spy* is almost unique in this. Most other screen based, interactive text editors have two modes; one for issuing editing commands and one for inserting text. Once you have entered INSERT mode commands have no effect. It can be frustrating to have to remember to move between modes before executing the next command. Also, leaving INSERT mode involves issuing another command and this slows down the task. The most natural way to edit text is to change the text wherever it is in the file and then move on to the next piece you want to change and *spy* provides this environment.

Spy is also unusual in that it allows you to edit several files at a time and to move or copy text between files as well as within a file.

Spy actually edits a copy of the file held in a buffer so although all changes are immediately effected it is only the copy that is changed so you can change your mind and finish an editing session without changing the original file.

Because *spy* is a point and act type editor and all the facilities are displayed on the screen, you must first understand the display before you can go ahead and edit. Once you understand the display, however, all action becomes obvious and detailed instructions become redundant. The following section is a tutorial introduction to using *spy*. If you are totally unfamiliar with *spy* you should work through the actions suggested in section 4.1 to gain familiarity with the editor. If you cannot work through the actions, the accompanying informative text is organised to be easily used for reference.

Spy assumes the default physical to logical button mappings (see Chapter 3 and *wprofile* (5)). Wherever a logical button is mentioned the default physical button is given in brackets. The Button Prompt area of the *spy* display also assumes the default mapping so it is a good idea to use these default settings. If using a different mapping you have to redesign the Button Prompt cursor (see Chapter 5) or do the required mental conversion for every action.

Section 4.2 is an editing ready reference summarising all the available facilities and necessary actions.

4.1 Using spy - a tutorial introduction

All the tutorial instructions are in bold type. Informative text is inset to further distinguish it from the tutorial instructions. Commands issued by selecting from the Command Menu or Control Box are always printed in capitals. Commands issued by a particular button press are printed with an initial capital letter but the rest of the command is in lower case.

4.1.1 Entering spy

Ensure that the Window Management System is running

Spy can only run in an existing WMS window. See Chapter 3 for details on running WMS and creating windows.

Select, or create and select, a large window to hold the *spy* display

The minimum window size depends on the font used. It must be possible to display at least three text lines in the editing area. Assuming the standard font the window needs to be at least postcard size, about two thirds of the width of the screen and one third of the height of the screen. If you try to invoke *spy* in a window that is too small, *spy* produces an error message in the window to tell you the window is too small by so many millimetres and draws a box to indicate the necessary size.

Its a good idea to use large windows until you are familiar with *spy*.

Make sure you leave the cursor in the selected window.

To edit a file using *spy*, simply type the *spy* command followed by the name of the file you want to edit. For example, to edit the file *test* you would type:

spy test

The *spy* display appears after a short delay, taking up the whole of the selected window.

Enter *spy* to look at any text file you have available on the PERQ

The file you want to edit may not immediately appear in the Text area of the *spy* display. This is because *spy* must first read the file and copy it into a buffer. Figure 4.1 shows you the *spy* display once a file is displayed and labels all the areas of the display.

Before the file is displayed, the Prompt area displays the prompt:

Reading file *filename*

As *spy* begins reading the display the background of the Prompt area inverts and then reinverts from left to right at a rate proportional to the rate at which *spy* is reading the file so you can see how long it is likely to take.

Small files are read almost immediately. A file of about 12 text pages still only takes 12 seconds before being displayed in the Text area.

4.1.2 The spy display

Spy divides the screen into seven different functional areas:

Prompt Area
Control Box
Search and Replace strings
Thumb Area
Command Menu
Scrolling Area
Text Area

} Editing area

All these areas are clearly labelled on Figure 4.1.

Try moving the puck around the tablet. Do not press any buttons on the puck just yet

Moving the puck moves a cursor on the screen and this cursor changes shape depending on the functional area to which it is pointing.

Move the cursor out of the *spy* window, if you haven't already done so by mistake, and back in again

The cursor reverts to the uncommitted Window Manager cursor. *Spy* cannot receive input from the tablet or the keyboard until you reselect the window for input from the tablet even though the window is still selected to receive input from the keyboard. Any keyboard input is queued until the window is reselected for tablet input.

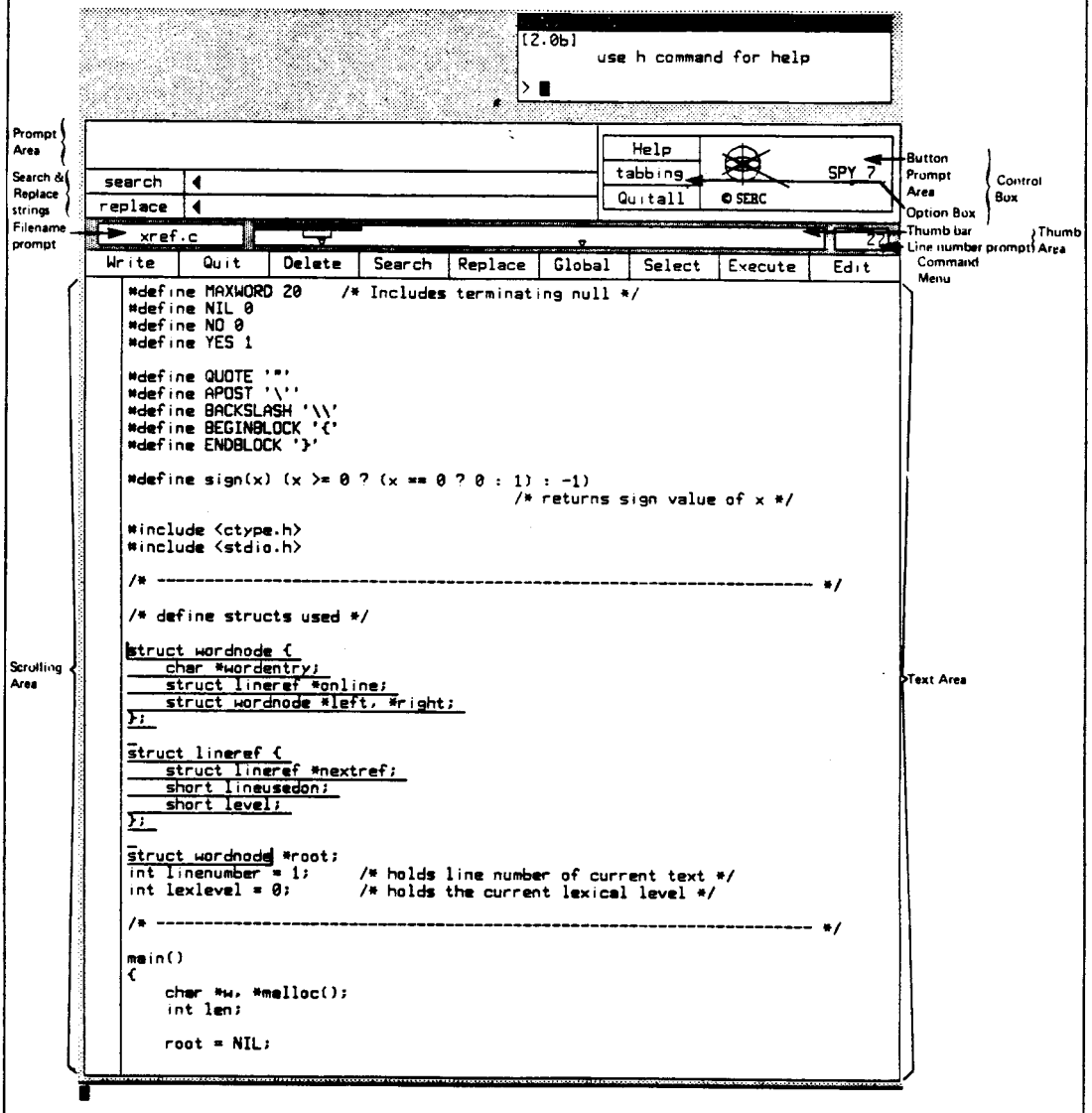


Figure 4.1 A typical SPY edit display

Press and release button A (left button or yellow button) to reselect the *spy* window

See Chapter 3 for a description of the Window Management System and an explanation of physical to logical button mappings.

Move the cursor into the Prompt area, over the Control commands, over the Search and Replace buffers, along the Thumb area, over the Command Menu, into the Scrolling Area and the Text area

You should notice that:

1 The cursor pattern changes

2 The Button Prompt area changes as the cursor pattern changes

3 The Line Number area changes when the cursor moves in the Text or Thumb area

The Button Prompt area displays the SERC logo whenever the cursor is in an area where button presses have no effect.

4.1.2.1 *Cursors*

Spy maintains four cursor shapes

1 The Cross Shaped Command cursor associated with the Command Menu and the Control Box is for selecting commands

2 The Hand Shaped Scrolling cursor associated with the Scrolling area is for scrolling forwards or backwards through the file

3 The Tuning Bar cursor associated with the Thumb area is for jumping to any part of the file

4 The Double T Selection cursor associated with the Text area and Search and Replace strings is for selecting text

4.1.2.2 *Prompts*

Cursor shape provides some feedback on the input *spy* expects. The Button Prompt area maintains a display to show you the effects of pressing one of the three puck buttons.

The Prompt area displays cues and messages. Whenever *spy* reads a file into a buffer for editing or writes from the buffer to a file, the area inverts and then reinverts from left to right at a rate proportional to the read or write to show you how long the action is going to take.

Some commands (EDIT, WRITE) require filenames. The Prompt area displays a prompt for the filename and any default filename. In this case the Prompt area is an editable area and keyboard input is directed to this area. You can type in a filename or edit the default in the usual way.

Consistency checks are done at various points and, if they fail, a message appears in the Prompt area which inverts to highlight the problem. Normally after about five seconds the message disappears and editor action continues.

4.1.3 Issuing commands

The cursor shape and Button Prompt Area give you feedback on the effect of a button press.

Pressing a puck button *selects* a particular action. Releasing the button *effects* the action.

Position the cursor over any command in the Command Menu and press but do not release the Command button

The background of the command inverts to show that it is selected.

Move the cursor into the Text Area and release the button

A message appears in the Prompt area to show you that the command is aborted.

Position the cursor over a command (not GLOBAL) and press but do not release the Command button. Move the cursor to point at another command and release the button

Again, no command is effected. You need to *press and release* to issue a command or effect an action. There is no time limit between the press and release.

Some commands require further action (QUIT, GLOBAL) in which case the suggested command is highlighted in inverse video.

To confirm a command just reissue it.

Table 4.1
spy commands

Command	*Effect*
EDIT	Creates another editing area and prompts for the file to edit
EXECUTE	Passes any selected text to the shell for execution. Any output is inserted in the file after the current selection and becomes the new selection
DELETE	Deletes the current selection
GLOBAL	Searches the current selection for all matches of the search string and either replaces or deletes them. Must be followed by the REPLACE or DELETE command
QUIT	Quits editing the file in that editing area. If this editing area is the last, quits *spy*. If the file has changed, QUIT prompts for you to WRITE the file
REPLACE	Replaces the current selection with the Replace string
SEARCH	Searches the file for a match to the search string and if it finds a match, selects it
SELECT	Selects the whole file
WRITE	Writes the file so that any changes are effected on the saved copy not thrown away with the buffer file

4.1.4 Help information

Issue the HELP command in the Control Box. Just move the cursor so that it is positioned over the command and press and release the Command button

This command displays general information on using *spy*. The HELP file is a text file listing all the information you need to use *spy*. This file is displayed superimposed over the whole Text area and you can scroll through it and edit it like any other text file (see section 4.1.6.1). While the HELP file is displayed you cannot edit your original file.

Get rid of the HELP file by issuing the QUIT command

If you alter a file, QUIT prompts you to WRITE the file first. Do not WRITE the HELP file. Just reissue the QUIT command.

Now try positioning the cursor over any of the commands in the Command Menu and press and release the Help button

In this case the help information is specific to the command. *Spy* searches the HELP file for the command selected and displays that part of the HELP file that describes the command.

Get rid of the help display by issuing the QUIT command

4.1.5 Options

Try positioning the cursor over the middle command in the Control Box and look at the Button Prompt area

This second command is in fact a circular option menu.

Press and release the Next button several times in succession to quickly view all the available options

Spy provides nine options with the following effects:

1 TABBING If on, then tab stops are automatically set at every eight spaces. Pressing the **TAB** button or ^I moves you to the next tab stop. If tabbing is off, then the **TAB** button or ^I displays the tab character (inverted ^I) on the screen and does not convert it to the required number of spaces

2 FULL FONT If on, then all characters including format and control characters are displayed exactly as the font represents them. If off, the end of file marker characters are visible and control characters are shown in inverse video

3 A=a Case sensitivity. If on, *spy* ignores the case of letters when searching for matches. If off, *spy* discriminates between upper and lower case letters

4 SILENCE If on, *spy* inhibits the audible beep which normally signals an error. If off, a beep alerts you to any problem

5 SCROLLING There are two scrolling speeds to determine how quickly *spy* updates the screen. SLOW causes *spy* to update line by line. QUICK causes spy to update the screen in small chunks. These speeds affect what you see when copying and moving text or moving to another place in the file

6 RAW INPUT This option is for inputting the special characters not available in the font set, for example, line control characters or printer control characters. RAW allows input of any characters in the first half of the ASCII character set. RAW + 0200 sets the top bit in characters to make the second half of the ASCII set visible. Characters with the top bit set are displayed with a diagonal dotted line through them to differentiate them from RAW INPUT. When RAW is off only the normal ASCII characters may be displayed

7 PATTERNS When the PATTERNS option is on, all the normal PNX metacharacters have their wild card value and may be used in the search and replace strings. The PNX metacharacters are documented in *spy* (1) and in the HELP file under SEARCH

8 LABELS Normally the Button Prompt area changes as you move the cursor to show the effect of a button press. If you find this distracting then you can turn LABELS off once you know how to use *spy*

9 WRAP If this option is set, SEARCH wraps past the end of the file to find a match

Try issuing the Toggle command for each option

The Toggle command changes the option value. The off state is represented by the option name with a diagonal line running through it. Notice how the FULL FONT and TABBING options change the appearance of your text file.

Try scrolling through the option list in either direction using the Next and Previous button commands

The options are included here so that you are aware of them but they are mostly for more specialist use.

There is a default setting for each option when *spy* is first invoked. The defaults are:

TABBING	on
FULL FONT	off
A=a	off
SCROLLING	slow
SILENCE	off
RAW INPUT	off
WRAP	off
LABELS	on

PATTERNS off

Some of these options may be set to the opposite value when spy is first invoked. The syntax is:

spy-tsfqerpl *filename*

t switches off tabbing, **s** switches off silence, **f** switches on full font, **q** turns on quick scrolling, **e** allows *filename* to start with -, **r** effects raw input, **p** switches on patterns and **l** switches off labels.

4.1.6 Traversing the file

Before you can begin to edit you need to find the piece of text you want to change. Initially *spy* displays the beginning of the file.

There are three mechanisms for finding what you want:

1 Scrolling

2 Thumbing

3 Searching

Searching is not covered in this section but is covered later.

4.1.6.1 *Scrolling*

Move the cursor into the marginal Scrolling Area beside the Text Area

Look at the Button Prompt Area to see how the puck buttons map onto the scrolling functions

Continuous scrolling

Position the scrolling cursor about halfway down the Scrolling Area. Press and do not release the Continuous button. Now move the cursor about one centimetre upwards

The displayed text moves upwards.

Move the cursor further up and notice how the scrolling speed increases as you move away and decreases as you move back towards the original cursor position. Now release the button

The original cursor position is marked with an inverted gloved hand. If you move the cursor up and keep the Continuous button pressed, the text in the Text Area starts to scroll up, the scrolling speed in proportion to the distance from the original position. If you move the cursor down from its original position keeping the Continuous button pressed, the text in the Text Area scrolls down, the scrolling speed in proportion to the distance of the cursor from its original position.

Jump scrolling

Position the scrolling cursor near the bottom of the Scrolling Area and press the Forwards button

Notice that the text has moved so that the line originally opposite the cursor is now at the top of the page.

Repeat the above jump. Now move the cursor so that it is a few lines from the top and press the Forwards button

You can control the size of the jump by deciding how many lines you want to move. When jump scrolling forwards through the file, place the cursor the required jump size from the top of the screen.

Reverse the above actions by placing the cursor the desired distance from the top of the screen and press and release the Backwards button

In jump scrolling, a continuous press on the jump buttons only causes one jump. You have to successively press and release the jump buttons to cause successive jumping.

The best way to anticipate the effects of jump scrolling is to visualise the Text Area as a window over the file. The forwards command moves the window towards the end of the file, the new top line of the window being the line opposite the cursor. The backwards command moves the window backwards towards the beginning of the file with the old top line becoming the new line opposite the cursor.

4.1.6.2 *Thumbing and marking text*

Move the cursor to the shaded Thumb area

In this area the cursor becomes a vertical oval shape with a broken vertical line running through it. This broken vertical line acts like the tuning bar on a radio.

There are three boxes in the Thumb area all to help remind you where you are. The left hand box displays the filename of the file you are editing. The right hand box displays the line number of the line pointed to by the cursor when it is in the Text area or the line number pointed at by the tuning bar cursor.

The central box is the thumb bar.

Move the cursor backwards and forwards along the thumb bar and look at the line number box. Notice how the line number changes

The cursor tunes in to a certain line in the file rather like the tuning bar on a radio tunes in to a certain frequency.

The thumb bar represents the whole file. A small black area on the upper part of the thumb bar indicates the proportion of the file actually on display in the Text area and indicates where the displayed text is in relation to the whole file. In a large file, the black cue bar is likely to be very small. For a file of one character, the black cue bar runs the length of the thumb bar. If the file is empty then the black cue bar is not displayed.

Try scrolling quickly forwards and backwards through the file to see how the black cue bar changes position

The thumb bar allows you to go to any part of the file in much the same way as you can turn to any page of a book.

Position the cue cursor near the end of the thumb bar and issue the Goto command

Spy displays the tuned in line in the middle of the Text Area and then fills in the screen below then above this line. This is to provide sufficient context for the tuned in line. This is particularly useful when editing source code for compiler errors. The compiler tells you the line number of the error. You can tune in to the error line using the thumb bar and then correct it. Source code errors are rarely solely on the line reported so the context is useful.

The lower area of the thumb bar can contain two cues; a selection cue and a marker.

Some part of the file or the search and replace strings is always selected ready for some editing action. When you first enter *spy* the beginning of the file is selected. The thumb bar also shows you where the selected text is. The selection marker is a small vertical line. If a section of text is selected, the selection marker becomes a horizontal line with vertical line delimiters. The length of the selection marker is proportional to the proportion of the file that is selected.

Position the cue cursor in the centre of the black cue bar and issue the Mark command

The black cue bar represents the current display. Pressing the Mark button *marks* this display with a black downpointing triangle on the lower part thumb bar so that you can return to it easily.

Position the cue cursor at the beginning of the thumb bar and issue the Goto command

The beginning of the text file is now on display.

Return to the marked display by placing the cue cursor over the triangle and issuing the Goto command

The display you were looking at near the end of the file should now be on display again.

Remove the mark by placing the cue cursor over the mark and issue the Unmark command

You can have up to 20 marks on the thumb bar at any one time. The advantage of the mark is that it marks the text not the line number so if you go back and delete text earlier in the file you can still return to the marked text even though the line number will have changed. In fact, the mark actually moves on the thumb bar to show that the marked text has moved its relative position in the file. Marks also travel with the text when it is moved or copied. In changing several compiler errors it is a good idea to mark all the errors first and then go back and correct them.

Return to the beginning of the file again using the thumb bar

You can go to the beginning or the end of the file by issuing the Goto command when the tuning bar cursor is beyond the left or right end of the thumb bar respectively.

4.1.7 Selecting text

Move the cursor into the Text Area

In this area the cursor is shaped like a broken I or two T's, one of them upside down. This is the selection cursor.

All the editing commands act on the current selection. You can select a position in the file or select an actual chunk of text.

Choose a paragraph of text for selection

Position the selection cursor at one end of the paragraph

Press and hold down the Select button and move the cursor to the other end of the paragraph

Release the puck button

Selected text is marked by underlining, with a vertical bar to mark each end of the selection, for example, |*selected*|.

Look at the thumb bar to see how the selection cue represents the selected text

When you issue the Select command, pressing the puck button fixes one end of the selection. The other end is dynamic until you release the button.

Scroll forwards through the file until the selected text is off screen

Choose another paragraph to select

Select one end of the paragraph by pressing and releasing the Select button

Issuing the Select command without moving the puck selects a position between characters rather than a piece of text.

Move the cursor to the other end of the selection and press and release the Extend button

Pressing the Extend button indicates the second end of the selection. This second end is dynamic until you release the Extend button.

Text in the Search and Replace buffers may be selected in the same way as text in the Text area

Position the selection cursor within the selection a couple of words from one end

Reissue the Extend command

You can extend to within a selection. *Spy* moves the end of the selection from the nearest end.

Position the selection cursor near the other end of the selection but within the selection

Issue the Extend command

Notice that it does not matter which direction you want to extend to or from, *spy* always shortens the selection by the least amount.

4.1.8 Amending text

Spy provides the following amendment possibilities:

Inserting text

Deleting text

Copying text

Moving text

Replacing text

Capturing output from PNX commands

All the above commands work on the current selection.

4.1.8.1 *Inserting text*

Type a few words on the keyboard

The words you type are inserted as you type to the left of the selected text. Selected text is pushed to the right as you type.

This is the beauty of a modeless editor. To insert text, all you need to do is select the position to begin the insertion and then just type your insertion.

The following characters have special effects:

RETURN	takes a new line
LF	takes a new line and indents input in line with the text above
^W	deletes the previous word
DEL	deletes the previous character
^U or **OOPS**	deletes the whole line
^R	redraws the screen completely

RAW INPUT allows you to enter these characters without any special effect.

4.1.8.2 *Deleting text*

Select the text you have just inserted

Delete your insertion by issuing the DELETE command in the command menu

The DELETE command always deletes the selected text.

DELETE does not bother to ask you to confirm that you really want to delete the selected text. Once you have deleted text you cannot retrieve it.

4.1.8.3 *Copying text*

Select a whole paragraph of text. Move to the end of the file using the thumb bar. Position the cursor before the end of file marker and press and release the Copy button

Spy copies the selected text to the new cursor position at the end of the file. The copied text is automatically selected ready for some further action.

4.1.8.4 *Moving text*

Return to the beginning of the file and position the cursor at the beginning of the file. Press the Move button twice in quick succession

A double press on the Copy/Move button not only copies the selected text to the new position but also deletes the original text. This is the way *spy* moves text.

The paragraph you copied to the end of the file was the selected text. This paragraph should now be the first paragraph in the file.

Return to the end of the file to check that the copied paragraph has now been deleted

Return to the beginning of the file and delete the first paragraph

4.1.8.5 *Replacing text*

Any selected text can be replaced by the text in the Replace buffer.

Position the cursor at the beginning of the Replace string area, after the word Replace but before any end of file marker

Issue the Select command

Type a sentence

Select a sentence in the text file

Copy it to the beginning of the file

The copied sentence becomes selected.

Replace the copied sentence with the Replace string by issuing the REPLACE command from the Command Menu

The REPLACE command is often used with some sort of search for text. Searching for text is covered later.

4.1.8.6 *Capturing text*

The EXECUTE command passes the selected text to PNX for execution. Any output is inserted in the file after the selection and becomes the new selection. This command can be used to copy files.

Insert the command:

cat *filename*

where *filename* is another file in your current directory

Select the text you have first typed

Issue the EXECUTE command

Delete the result of the EXECUTE command by issuing the DELETE command

4.1.8.7 *Quitting without updating*

Issue the QUIT command

If you have changed the file you are editing QUIT reminds you to WRITE the file by highlighting the WRITE command.

Reissue the QUIT command to confirm it

Normally you would issue the WRITE command to effect the changes and then QUIT.

4.1.9 Creating text

Reenter *spy* giving the command:

spy

with no filename.

If you do not give a filename. *spy* assumes you are going to create a file.

Position the selection cursor at the beginning of the new Text area and type the following paragraph:

> **Fred and Mary went to Scarborough for the holidays. Unfortunately, when they got there they found that their rooms in the Hotel Splendide had been let to someone else. Fred was very annoyed but there was nothing for it. They would have to start looking for another hotel**

You have now created some text. The following sections assume that you are editing this text, called TUTORIAL.

4.1.10 Searching for text

There are two searching commands. The SEARCH command looks for a match for the Search string and, if it finds it, selects it ready for you to do whatever you like with it; change it, replace it or delete it. The GLOBAL command looks within the selected text for every match of the Search string. The GLOBAL command should be followed by another command, REPLACE or DELETE, to tell *spy* what to do with all the matches but you can abort the GLOBAL command by issuing any other command.

Move the selection cursor to the Search string area and position it before the end of file marker

Type Hotel Splendide

Hotel Splendide is now the Search string and when you issue the SEARCH command *spy* searches from the selection for this string and if it finds it, selects it.

Select the beginning of the file TUTORIAL and issue the SEARCH command

Spy searches forwards or backwards through the file from the selection. When you press to select the SEARCH command, the SEARCH box changes to show two prompts; < on the left and > on the right. Releasing the button while the cursor is on the left prompt starts a backward search. Releasing the button while the cursor is on the right prompt starts a forward search.

Position the selection cursor in the Replace string area in the TUTORIAL editing window before the end of file marker

Type George and Dragon

George and Dragon is now the Replace string and when you issue the REPLACE command *spy* replaces the selected string with the Replace string.

Issue the REPLACE command

Fred and Mary should now be annoyed with the George and Dragon rather than the Hotel Splendide.

Replace the Search string by selecting and deleting the old search string and then insert Fred

Replace the Replace string by selecting and deleting the old Replace string and then insert Albert

Select the whole of the file TUTORIAL by issuing the SELECT command from the Command Menu

The SELECT command selects the whole of a file.

Issue the GLOBAL command and hold down the button

The GLOBAL command searches the whole of file TUTORIAL for all matches to the Search string for deletion or replacement. GLOBAL requires additional input. When you press the button GLOBAL becomes shaded to show that it is selected and DELETE and REPLACE become highlighted to show that either one may be selected. Move the cursor to either of these commands before releasing. If you release the button in any other area the GLOBAL command aborts.

Issue the REPLACE command by releasing the button with the cursor over REPLACE

Notice that *spy* replaces all instances of Fred with the Replace string. Now Mary has a new partner having swapped Fred for Albert.

4.1.11 Ending an edit session

If you want to save the file TUTORIAL for posterity then perform the following action:

Issue the WRITE command in the TUTORIAL Command Menu

WRITE prompts you for the name of the file but displays the full path name that you have already given it.

You can confirm the displayed pathname by issuing WRITE again or replace it as follows:

If you want to delete the whole pathname, press OOPS

Delete only the old filename by repeatedly pressing the DEL key until you are satisfied with what is displayed. The DEL key erases from right to left, character by character

Type the new pathname or filename followed by RETURN then issue WRITE again to confirm

WRITE also inverts the Prompt area and then reinverts it at a rate proportional to the WRITE to give you some idea of how long the WRITE will take.

Issue the QUITALL command in the Control Box

Spy reminds you to WRITE files if you have not already done so and highlights the WRITE command in the Command Menu of any unwritten file.

You do not have to write the file.

If necessary confirm the QUITALL command by issuing it again

You should now be returned to PNX shell.

4.1.12 Multiple file editing

Spy can edit more than one file at a time, the limit depends on the amount of editing space available (see section 4.1.1).

Reenter *spy* with any existing file as the parameter

Issue the EDIT command in the Command Menu

Spy prompts you for the name of the file you want to edit. To edit an existing file you should give the name of the file and reissue the edit command.

Just issue the EDIT command again

If you do not give a filename and reissue the EDIT command, *spy* assumes you are going to create a file and creates an empty buffer.

Spy divides the Text area equally between the file you were editing and the new file.

Each editing area includes a Thumb area, Command Menu, Scrolling area and a Text area.

Try issuing the Edit command in the new empty editing area. Select EDIT again in response to the filename prompt

Spy divides this Text area equally.

Delete the central editing area by issuing the QUIT command from the Command Menu in the central editing area

Spy gives any free space to the top editing area.

You can get a multiple file display by issuing EDIT commands from within the display. Alternatively, when you first invoke *spy* you can give more than one filename as a parameter to *spy*, for example:

Spy file1 file2 file3

would divide the available editing area between the three files.

The minimum size of a Text area is about three lines of text. *Spy* can create up to 6 editing areas with this restriction. Figure 4.2 shows a typical multiple file display.

Select a paragraph in the top editing area

Move the cursor to the beginning of the Text area in the second editing area

Copy the selected paragraph to the new file

Spy allows you to copy and move text between files in one *spy* display as well as copy and move text within a file.

Issue the QUIT command from the Command Menu in the lower editing area

If a file changes, QUIT prompts you to WRITE the file. A repeat of the command confirms the QUIT.

Delete the area by reissuing the QUIT command

You have now tried every facility that *spy* has to offer and should be fairly familiar with the display.

Spy provides sufficient cues and messages so you should not need to refer to this publication again. However, section 4.2 provides a ready reference for all the editing facilities and you can check how to do things by looking at the relevant part of this tutorial.

4.2 Editing ready reference

Figure 4.3 provides a diagrammatic reminder of the appearance of the cursor in the different functional areas of the *spy* display and reminds you of the effect of pressing any of the puck buttons in that area.

Table 4.2 provides an alphabetic list of editing functions together with instructions for performing the required action. Any underlined actions are described in this table.

Table 4.2
Spy functions

Function	Required actions	Comments
Append text	1 Find the end of the file 2 Select the position after the last character 3 Type the text	There is no difference between appending text and inserting text
Copy text	1 Select the text to be copied 2 Move the cursor to the required position for the beginning of the copy 3 Press and release the Copy button	You can copy text to a new place within the file or to another file displayed in another editing window in the same *spy* display

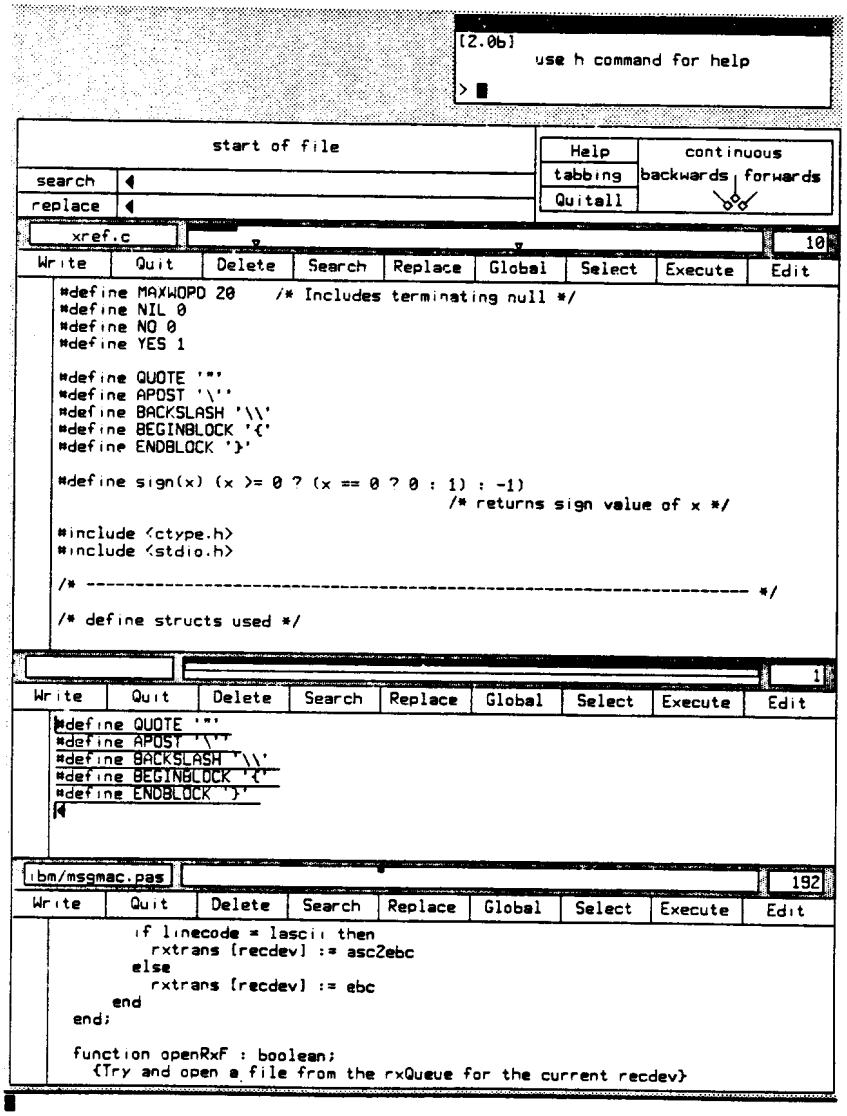


Figure 4.2 A multiple edit display

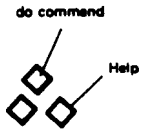
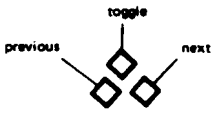
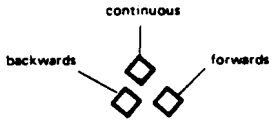
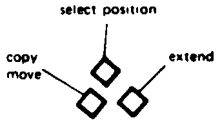
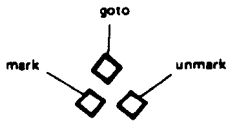
Cursor		*Button effects*
Cross shaped Command cursor 	Command Menu Control Box Command Menu	
Cross shaped Command cursor 	Option box in Control Box Command Menu	
Hand shaped Scrolling cursor 	Scroll area	
Double T shaped Selection cursor 	Text area Search and Replace strings	
Tuning bar Cue cursor 	Thumb bar	

Figure 4.3 Spy cursors and puck button effects

Table 4.2 (continued)

Function	Required actions	Comments
Create a text window	1 Issue the EDIT command 2 Type the filename of the file to be edited or created 3 Issue EDIT again	The prompt for the filename appears in the prompt area
Create a text file	1 Create a text window 2 Move the cursor to the beginning of the file 3 Type the text	
Delete by character	1 Select the position after the character 2 Press the **DEL** key	
Delete by	1 Select the position at the end lineof the line 2 Press the **OOPS** key	
Delete text	1 Select the text 2 Issue the DELETE command	
Delete a text window	1 Issue the QUIT command 2 Issue QUIT again if reminded	The QUIT command reminds you to WRITE the file if you have made changes
Deselect text	1 Select another piece of text	There is always a selection
Ending an edit session	See Exit *spy*	
Enter *spy*	1 Type the *spy* command followed by any options you want to set followed by the filenames of the files you want to edit	You can give *spy* up to 6 filenames as parameters and *spy* creates a Text area for all of them if there is room in the window manager window. The filenames should be separated by spaces
Exit *spy*	1 Update the files you want to update 2 Issue the QUITALL command for multiple file editing or issue the *QUIT* command if there is only one editing area 3 Confirm the command if necessary by issuing it a second time	QUITALL reminds you to update files you have been editing if you have not already done so. You need to update the files and reissue the command to confirm

Table 4.2 (continued)

Function	Required actions	Comments
Find text	See Scroll, Search and Thumb	
HELP information	1 Issue the HELP command in the Control Box	This gives general information. The information is output over all Text areas, covering all files
	2 Position the cursor over a command and press the Help button	This gives specific Help information on that command
Insert text	1 Select the position to begin the insertion	There is no separate mode for text insertion
	2 Type the text	
Issue commands	1 Move the cursor to a position over the required command	Table 4.1 describes the effects of the commands in the Command Menu
	2 Press the Do command button	
Mark displayed text	1 Position cursor in the centre of the black cue bar representing the displayed text on the thumb bar	Marks are tied to actual text so even if the line number changes you can get back to the text you want
	2 Press the Mark button	
Mark line number	1 Move the cursor along the thumb bar until the required line number is shown in the line number box	This is particularly useful for correcting compiler errors in source code. You can make up to 20 marks
	2 Press the Mark button	
Merge text	1 Enter *spy* with both files as parameters or issue the EDIT command to get both files displayed in their own Text areas	Merging text is exactly the same as moving text within a file
	2 Select the text to move	
	3 Move the cursor to point to the required position for the insertion	
	4 Press and release the Move button twice in quick succession	
Move text	1 Select the text	You can move text within a file or between files displayed in different editing windows of the same *spy* display
	2 Move the cursor to the required new position	
	3 Press and release the Move button twice in quick succession	

Table 4.2 (continued)

Function	Required actions	Comments
Options	1 Position the cursor over the option box 2 Press the Previous or Next puck button successively to move through the option list to find the option you want to change 3 Change the option state by pressing the Toggle button	Figure 4.1 labels all the parts of the *spy* display. The effect of each option is described in section 4.1.5 and documented in the HELP file The Off state is represented by the option name with a diagonal line through it
Remove a Mark	1 Place the cursor over the mark on the thumb bar 2 Press the Unmark button	
Replace text	1 Insert the Replace string in the Replace string area 2 Select the text to be replaced 3 Issue the REPLACE command	When doing a Search and Replace the text is selected for you
Return to	1 Position the cursor over the marked textmark on the thumb bar 2 Press the Goto button	
Scroll through text continuously	1 Move the cursor into the Scroll Area 2 Press and hold down the Continuous button 3 Move the cursor in the direction you want the text to scroll. Move up to scroll up or forwards through the text. Move down to scroll downwards or backwards through the text 4 Release the Continuous button to stop scrolling	Speed of scrolling is proportional to the distance you move the cursor away from its original position. The original position is marked by an inverted cursor shape
Scroll through text in jumps	1 Move the cursor into the Scroll Area 2 Place the cursor so that it is the size of the jump you want away from the upper or lower line 3 Press the Forwards button to jump text up 4 Press the Backwards button to jump text down	

Table 4.2 (continued)

Function	Required actions	Comments
Search for one piece of text	1 Insert the text to be searched for in the Search string area 2 If you cannot remember the case of the text you are searching for change the case sensitivity option so that it is turned off 3 SEARCH searches from the selection in the required direction so select text in the appropriate Text area 4 Issue the SEARCH command	*Spy* automatically selects the searched for text when it finds a match When you press the button to issue the SEARCH command, the SEARCH box displays the direction alternatives; < on the left indicates backwards, > in the right indicates forwards. Releasing the button in either half of the SEARCH box starts the search in the pointed at direction
Search for all matches to text to replace or delete	1 Insert the text to be searched for in the Search string area 2 Turn off the case sensitivity option 3 Select the text to be searched 4 Issue the GLOBAL command and keeping the button down move to issue the DELETE or REPLACE command to change all matches	GLOBAL finds every match and replaces or deletes the matches. Abort the command by releasing the button in any area except REPLACE or DELETE
Select a position	1 Move the cursor to the position you want 2 Press the Select button	This is used for inserting text. Selecting a position marks the text with a vertical bar
Select text	1 Select a position at the beginning or the end of the desired selection 2 Move the cursor to a position at the other end of the selection 3 Press the Extend button	Selected text is underlined and marked at either end by vertical position markers
Select the file	1 Issue the SELECT command	This command is useful for the GLOBAL command and for copying the whole file to another

Table 4.2 (continued)

Function	Required actions	Comments
Thumb through text	1 Position the cursor on the thumb bar at the place corresponding to the place in the file you want 2 Press the Goto button	The thumb bar represents the whole of the file. A black cue bar represents the amount of the file currently in display in the Text Area. While the cursor is on the thumb bar the line number window shows you the line number you would thumb to. The thumbed to line is displayed in the centre of the resulting text display
Update a file	1 Issue the WRITE command 2 Change the filename if necessary by deleting the old filename and typing the new filename 3 Issue WRITE again	*Spy* prompts for the filename of the file to be written and displays the current name in the Prompt area. You can delete this by pressing **OOPS** or delete it character by character by pressing **DEL.** Type the new file name and press **RETURN**

5.1 Cursors

The PERQ hardware cursor is a small picture on the screen that usually, but not necessarily, tracks the movement of the pointing device on the tablet. The cursor pattern is not placed on the bit map but superimposed on the screen at each screen refresh. A particular coordinate in the cursor pattern points at the screen (or window) coordinate in the underlying bit map so a process receiving input from the tablet knows the item being pointed at, for example, the actual screen coordinate for graphics input, a menu item for command selection or a particular window for window selection.

PNX cursors have the following properties:

1 PATTERN A 57 bit by 64 bit picture within a 64 bit by 64 bit area. Pictures may be animated by programs

2 ORIGIN The coordinates of the bit in the cursor pattern area that actually points at the screen. The origin does not need to be within the cursor pattern area. The origin may change in an animated cursor

3 FUNCTION The way in which the cursor pattern combines with the underlying screen bit map

4 MODE The way in which the cursor tracks the movement of the puck

5 POSITION Normally the cursor tracks the pointing device but the tracking function may be turned off

Cursor files contain the pattern and origin information. Cursor properties, that is, the cursor file in use, the cursor function mode, and position are a subset of window properties. The initial cursor file, function and mode to be used in a particular window may be set in the window description file using *mkwind*(1). *mkwind* places the information in the structure *wdesc* (see *wdesc*(5)).

Cursor properties may be changed from within a program by *ioctl* calls (see section 5.2 and *window*(4)).

5.1.1 Cursor files

Cursor patterns may be redesigned, copied or created using *cedra* (see section 5.3). Cursor files may also be transferred from POS machines.

A cursor pattern is 64 bits high and 64 bits wide but the actual picture may only be up to 57 bits wide.

The origin, the actual bit in the cursor pattern that points at the screen, is specified as an X and Y offset from the top left corner of the pattern. The top left corner is the default origin (0,0). Note that the origin is not constrained to be within the cursor pattern area.

A PNX cursor file requires at least 516 bytes of store, 512 bytes for the 64 x 64 bit pattern and 4 bytes for the integers (two short integers) to designate the origin. An animated cursor file consists of multiples of 516 bytes, one for each pattern in the cycle. PNX also accepts a cursor file of 512 bytes (POS type cursor). PNX assumed a default origin of (0,0) for such files.

When reading in cursor files, store can be allocated in several ways:

1 Declaring a variable of type union *sgwinb* (see *window* 4)). This allocates space for all cursor properties (except the pattern)

2 Declaring a variable of type (see *wcurs*(5)):

```
struct wf__cpat {
                cur__pat pattern:
                short    CuPX;
                short    CuPY;
};
```

cur__pat is declared in *sgwin.h* as:

```
typedef unsigned int     cur__pat[128]
```

3 Allocating space using *malloc*(3)

4 Allocating space using *rectangle*(3)

Animated cursors can be allocated space by declaring an array of the structure as defined above.

The structure *w_cpat* in *sgwinb* points to a cursor in store but when declared alone does not actually allocate store. An animated cursor could be declared as an array of this structure:

```
struct w__cpat cursor[N]
```

in which case each cursor pattern, cursor [0] to cursor [N], must be allocated store using *malloc*(3) or *rectangle*(3) (see section 5.2). Use *lseek* to find the size of the file before using *malloc* or *rectangle*.

Figure 5.1 demonstrates the appearance of all the cursors provided with PNX and tells you where these cursor patterns are stored.

POS cursor files do not contain an origin so are in multiples of 512 bytes. POS cursors can be copied to PNX. If PNX reads a cursor with no origin then the top left corner (0,0) is the default.

5.1.2 Cursor functions

The cursor function in use not only determines the way in which the cursor bit pattern combines with the underlying screen bit pattern but also determines the background colour of the screen or a window.

wprofile(5) determines the colour of the screen. If the screen is white then the normal background colour for windows is white. Cursor functions which invert the window background make the window black and the cursor pattern and window pattern white. If the screen is set to black, normal windows are black and inverted windows become white.

The cursor functions are integer constants defined as follows:

```
CFNORMAL        4   /* Normal screen or window, cursor superimposed, logical OR */
CFINVERT        5   /* Inverts screen or window, cursor superimposed, logical OR */
CFCURSCOMP      6   /* Normal screen or window, cursor opposite to background,
                       logical XOR */
CFINVCURCOMP    7   /* Inverts screen or window, cursor opposite to background, logical
                       XOR */
CFWHITEHOLE     3   /* Normal screen or window, cursor inverted and combined with
                       screen, logical AND */
CFBLACKHOLE     2   /* Inverts screen or window; cursor normal and combined with
                       screen, logical AND */
```

Figure 5.2 shows the effect of each of these cursor functions.

The union *sgwinb* (see *window*(4)) contains a field for the declaration of cursor function. Assuming that a variable *SG* has been declared to be of type *sgwinb*, for example:

sgwinb SG;

The field *sg__CuFunc* can be assigned any of the above cursor function values. The following *ioctl* call (see *window*(4)) sets the cursor function:

ioctl (1,WIOCCFUN, &(SG.sg__CuFunc))

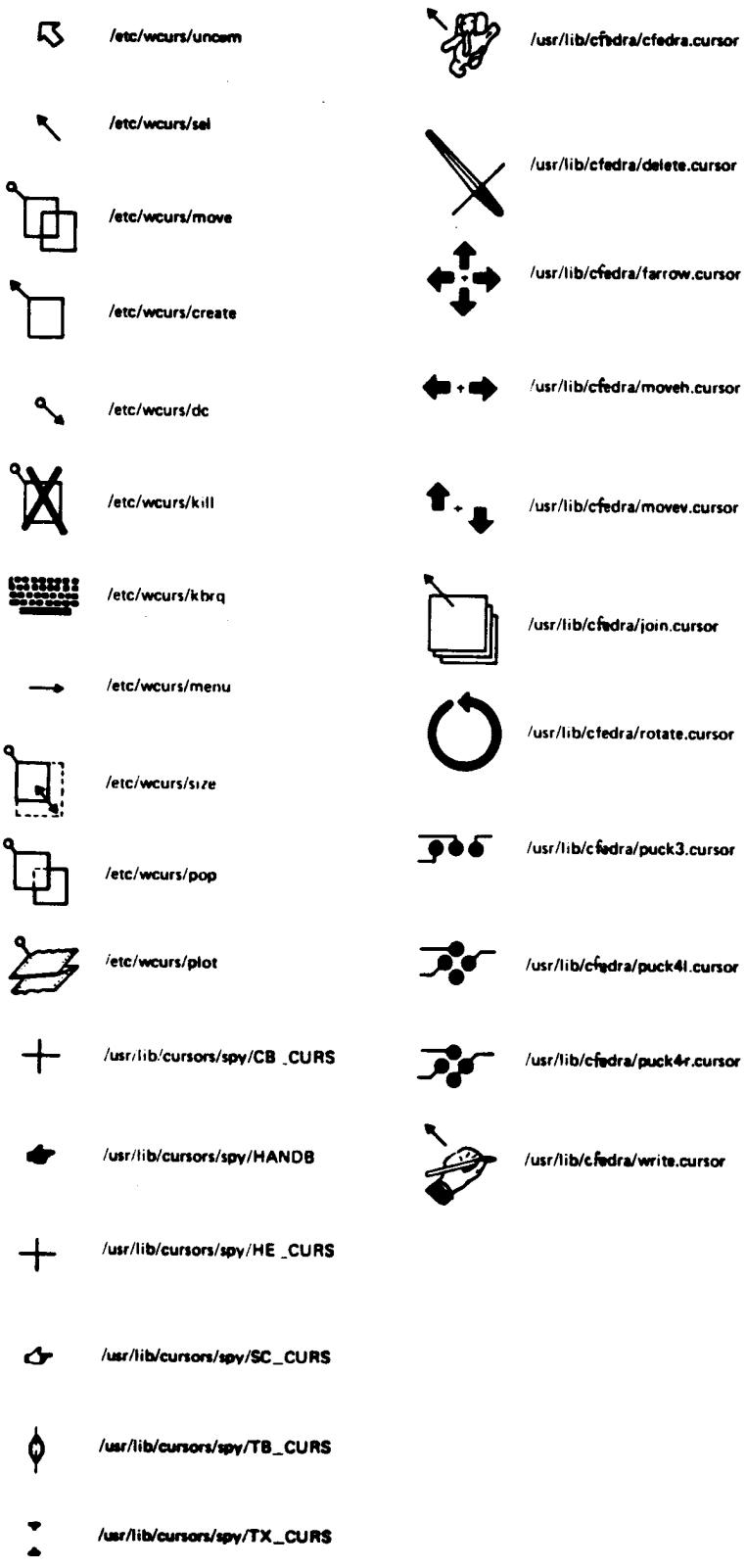


Figure 5.1 PNX cursor files

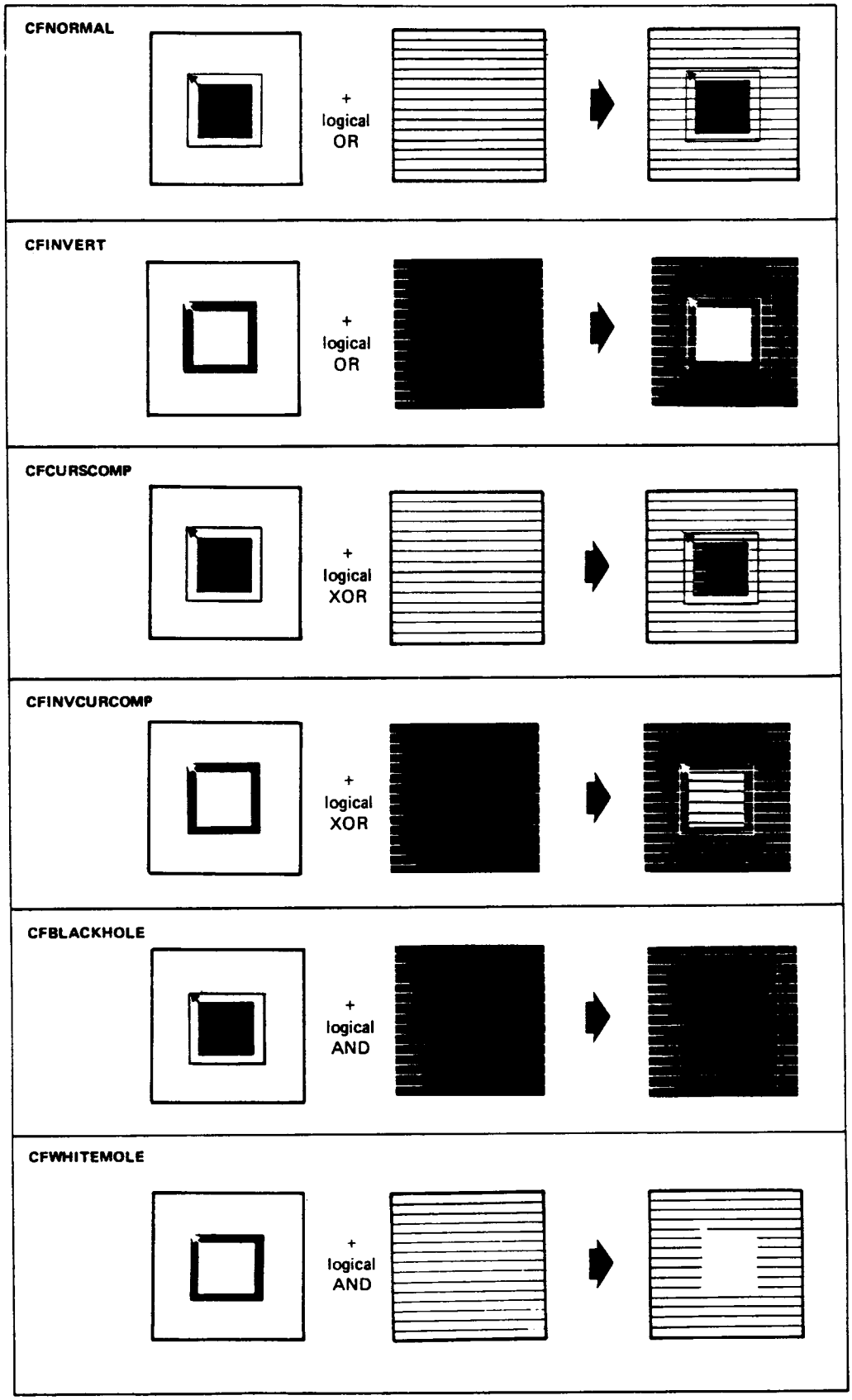


Figure 5.2 Cursor functions

5.1.3 Cursor modes

A cursor can track the pointing device on the tablet absolutely, that is, the coordinate of the puck on the tablet corresponds exactly to the screen coordinate pointed at by the cursor. A cursor can also track relatively. In relative tracking the cursor tracks the movement of the pointing device exactly but if the pointing device is removed from the tablet and then replaced in a new position then the cursor does not reflect the new position of the pointing device. If the pointing device is moved from its new position the cursor tracks the device in direction and length not position.

Whether the cursor tracks absolutely or relatively depends on the set up of the tablet (see *tablet*(4)).

Whether the cursor tracks relatively or absolutely, there are four cursor modes that sit on top of the basic tracking function. The four modes are integer constants defined as follows:

CMAUTO	0	/* tablet input directed to the window as long as cursor points within the window */
CMXAUTO	1	/* tablet input directed to the window wherever the cursor points */
CMON	2	/* cursor does not track the pointing device but appears at the position specified by the process. Moving the pointing device has no effect on the cursor */
CMOFF	3	/* cursor picture switched off. The cursor invisibly tracks the pointing device as in CMXAUTO. If any special cursor becomes enabled then it does become visible */

The union *sgwinb* (see *window*(4)) contains a declaration for the cursor mode. The variable *SG.sg__CuMode* can be assigned one of the above values. The following *ioctl* call (see *window*(4)) sets the cursor mode to the value of *SG.sg__CuMode.*

ioct(1,WIOCCMOD,&(SG.sg__CuMode));

5.1.4 Cursor position

When the cursor mode is CMON, a program can specify the position of the cursor relative to the window origin, that is, the top left corner of the window's usable area.

The union *sgwinb* contains a field sg__WPOS with the following structure declaration:

```
struct w__cpos {            /* bit coordinates of the cursor origin relative
                               to the window origin */
        short CuPOSX;
        short CuPOSY;
        };
```

Assuming a variable *SG* of type *sgwinb*, values assigned to the variables *SG.sg__CuPos.CuPosX* and *SG.sg__CuPos.CuPosY* may be set using the following *ioctl* call:

ioctl (1, WIOCCPOS, &(SG.sg__CuPOS))

5.2 Using cursors

5.2.1 Input mode

If a program is to use the cursor as an input device then the window must be selected for input from the tablet. The union *sgwinb* declares the variable **sg__*inMode*** with the following structure:

```
struct w__inm {
        char    InMode;
        char    TrFlags;
        short   TrTabs;
        int     TrTicks;
      };
```

See **wmndow**(4))0for the values the above fields can take. The variable (*SG.sg__InMode* specified the type of tablet input that is of interest and how the input record is to be treated.

Having declared values for the variables in *SG.sg__InMode* the following *ioctl* call sets the input mode for the window.

ioctl (1,WIOCENIN,&(SG.sg__InMode))

5.2.2 Special cursors

The cursor may be used to point at an item on the screen, for example, to select from a menu. The cursor may also be used for graphical input to the screen or window. PNX provides a special cursor for graphical input. The special cursor does not replace the standard cursor tracking the puck but is an additional facility. The field *sg__SpCu* is declared in union with *sgwinb* with the following structure.

```
struct w__cpu {
              short CuSType;
              short CuSSp1;
              short CuSP1X, CuSP1Y; /* coordinates of point 1 */
              short CuSP2X, CuSP2Y; /* coordinates of point 2 */
             };
```

There are four standard special cursors which are acceptable values for CuSType:

CSTFRAME	A fixed size rectangle. Points 1 and 2 are opposite corners of the rectangle. Their coordinates declared relative to the standard cursor origin
CSTRBOX	A rubber box. Points 1 and 2 are opposite corners of the rectangle. Point 2 is fixed at the standard cursor position when the special cursor is enabled. Point 1 moves relative to the standard cursor
CSTRLINE	A rubber line. Points 1 and 2 are the ends of the line. Point 2 is fixed at the standard cursor position when the special cursor is enabled. Point 1 moves relative to the standard cursor
CSTXHAIR	Cross hairs central on Point 1 which moves relative to the standard cursor

Even if the standard cursor mode is CMOFF, the special cursor tracks the cursor position as if the standard cursor were in CMXAUTO mode.

window (4) describes the full potential of the special cursor facility.

Assuming a variable *SG* of type *sgwinb*, the following *ioctl* call enables the special cursor:

ioctl (1,WIOSPCU,&(*SG*.sg__SpCu))

5.2.3 Programming examples

Setting a cursor pattern involves the following steps:

Allocating store

Reading cursor file into store

Setting cursor to the read in pattern

Example 1 Copies a cursor pattern

```
#include <error.h>
#include <stdio.h>
#include <sgwin.h>
#include <wuser.h>

#define LINESIZE 40

extern int errno;

main ()
{
int inmode, rept, s;
int op;
union sgwinb ioc;
struct wf__cpat pattern;

ioc.sg__CuPat.CuPatp = (int*)(patter.fCuPat);
if (op = ioctl(0,WIOGPAT,&ioc))<0)
      {
       if (errno != ENOTTY)
      }
            {
             perror ("GPAT ioctl error");
             exit (99);
            }

      /* assume disc file and read it in */
      if (op = read (0,(char *)&patter, sizeof (patter)))<0

            {
             perror ("read pattern");
             exit(99);
            }

      if (op < sizeof (cur__pat))

            {
             fprintf (stderr, "read pattern too small \n");
             exit (99);
            }

      if (op < (sizeof( cur__pat)+2))
      pattern.fCuPX = :                          /* set origin to (0,0) for
                                                    POS type cursor */
      if (op < (sizeof ( patter))
      pattern.fCuPY = 0;


      /* pattern now read in to ioc.w__cpat */
      if (op = ioctl (1, WIOCCPAT, &ioc)<0)
            {
             if (errno != ENOTTY)
             perror ("CPAT ioctl error");
             else
            {
             patter.fCuPX = ioc. sg__CuPat.CuPX;
             patter.fCuPY = ioc. sg__CuPat.CuPY;
             if (write (1, (char *)&patter, sizeof (patter)) <0
             perror ("write pattern");
            }
      }
}
```

Example 2 Displaying an animated cursor

```
#include <sgwin.h>
#include <wuser.h>
#include <stdio.h>

#define MAX 48

char*rectangle();
struct w_cpat cursor[MAX];
struct w_inm inmode = {IMSAMPLE,0,0,0};

main (argc, argv)                       /* the program expects two parameters - the
char *argv[];                           name of the cursor file and the pause
                                        (units 1/60 seconds) between each picture in
{                                       the cycle */
  int i = 0, j, k, pause = 4000;
  int fp, fsize, csize;

if (argc < 2)
      {printf ("usage: animate cursor [pause]\n");
       exit (1);
      }

if ((fp = open(argv[1], 0)) <0
      {printf ("Can't open %s\n", argv [1]);
       exit (1);
      }

fsize = lseek(fp, 0L, 2);               /* find size of file in bytes */
lseek (fp, 0L, 0);                      /* rewind */
if (fsize%512)
      csize = 516                       /* PNX type cursor */
else
      csize = 512                       /* POS type cursor */
ioctl (1, WIOCENIN,&inmode);
if (argc>2) pause = atoi (argv[2];
do
      {
           cursor [i].CuPatp = (int *) rectangle (512);
           read (fp, (char *) cursor [i++].CuPatp, 512);
           fsize -= 512;
           if (csize == 516)
                 {
                    read (fp, (char *)(&(cursor [i-1].CuPX)),4);
                    fsize -= 4;
                 }
      } while ((fsize) && i<MAX);

for (;;)
      for (j = o, j<i; j++)
      {
        ioctl (1, WIOCCPAT, &cursor [j]);
        for (k=o; k <pause;k++);
      }
}     /* end of main */
```

5.3 Cursor editing

Cedra (Cursor EDitor for RAster patterns) allows you to create or modify the raster patterns for cursors. Like *spy*, the PNX text editor, *cedra* is a screen based, interactive, point and act type editor. Almost all interaction is through presses on the puck buttons.

When invoked, *cedra* establishes a window the full size of the screen (except the echo area) and, like *spy*, divides the window into several functional areas. Moving the puck on the tablet moves a cursor on the screen. The cursor may change shape as it moves into a new functional area.

Within each functional area, presses on each of the puck buttons have a different effect so an information area, known as the *mousehole* (see Figure 5.3), reminds you of the effect of a button press. A further information area describes the current *environment*. The environment corresponds to the type of input *cedra* expects. Environments mostly correspond to functional areas but each functional area may have more than one environment.

Since all editing is through the display, once you are familiar with the display using *cedra* becomes self explanatory. Section 5.3.1 describes the display in detail. Section 5.3.2 describes using *cedra*.

5.3.1 Cedra display

Figure 5.3 shows the *cedra* display with the functional areas and information areas clearly labelled. These areas are described below:

5.3.1.1 *The mousehole*

The mousehole maintains a pictorial representation of the puck buttons, labelling each button with the current effect of pressing that button. An asterisk beside the button representation indicates that pressing that button has no effect.

cedra looks to see what type of tablet is connected as */dev/tablet* and displays either a three button or four button picture. *cedra* then checks the logical to physical button mapping in *wprofile*(5) to see whether the picture should be labelled as a right hand or left hand puck. A right hand puck ensures that logical button A is physical button 0. A left hand picture assumes that either logical D or logical C is physical 0. *cedra* works with the following mappings:

Four button puck		*Three button puck*
Right hand		*Right hand*
A = 0		A = 0
B = 1		B = 1
C = 2 or 3 }	either may be	C = 2 or D = 2
D = 2 or 3 }	omitted and C or D set to 2 and 3	
Left hand		*Left hand*
A = 0		D or C = 0
C = 1 or 2 }	either may be	B = 1
D = 2 or 1 }	omitted and C or D set to 2 or 1	A = 2
B = 3		

In certain circumstances the program requires input from the keyboard, for example, the filename of the file to be edited. On these occasions all buttons on the puck are marked as unavailable and an explicit prompt for keyboard input appears in the message area.

5.3.1.2 *Environment area*

The current *environment* may be defined as the effect an input would have at any one time. Each functional area on the screen represents a particular environment since button presses within each functional area have an effect dependent on the area.

Within a functional area, you may in fact select environments depending on the button press, each button press resulting in a different environment.

Initially there are three environments corresponding to the following functional areas:

Select	Picture areas
Edit	Grid area
No command	Any area apart from the above

A pop up menu of commands is available in any area except the picture areas. This changes the environment to Menu. Issuing any menu command changes the environment to the environment for that command.

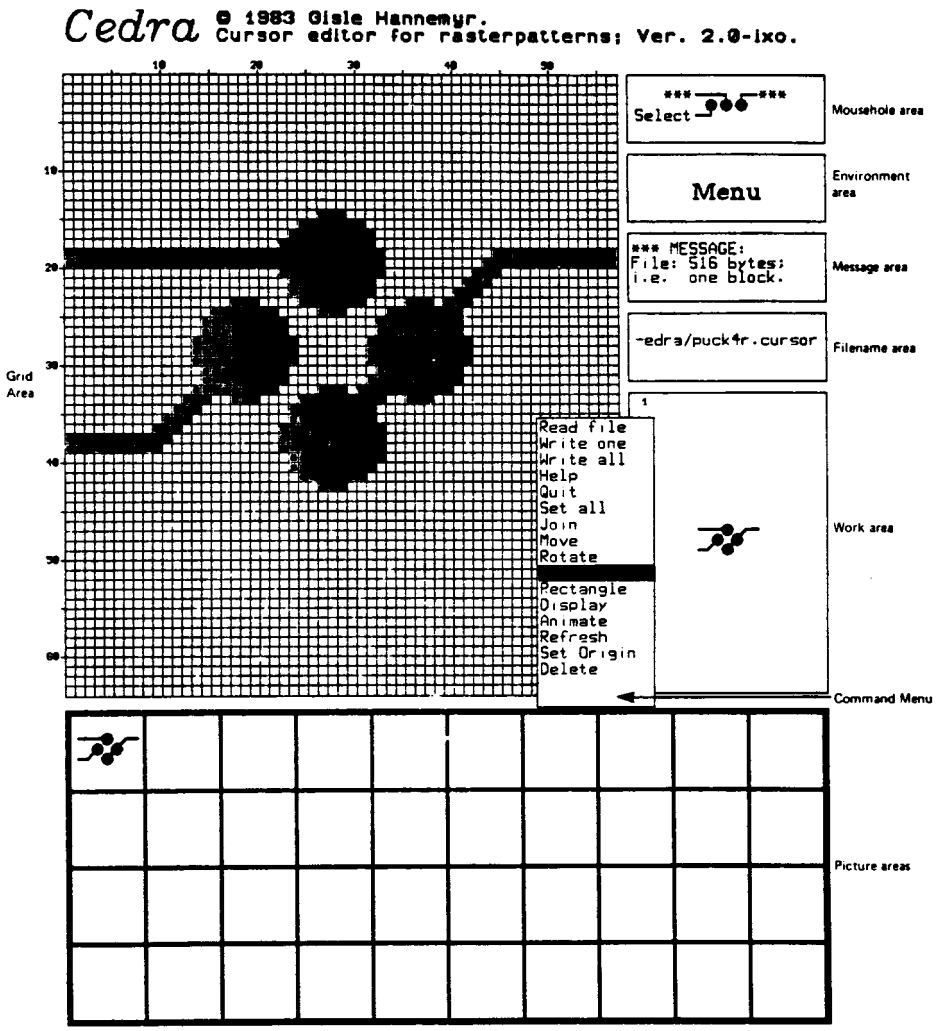


Figure 5.3 A typical Cedra display

5.3.1.3 *Message area*

This area displays messages and prompts from *cedra* and echoes any typed response.

5.3.1.4 *Filename area*

This area shows the name of any cursor file being edited in the grid area.

5.3.1.5 *Grid area*

Initially the grid area is blank with a vertical and horizontal scale along the outer edges. Initially, the message area prompts for the name of the cursor file to edit. Either type the filename followed by **RETURN** or just press **RETURN**.

Once you have specified a cursor file to edit, the area becomes a grid display with an enlarged representation of the cursor. The representation is to scale, each square on the grid representing a pixel on the PERQ screen. If you specify a cursor file that cedra cannot find, *cedra* displays a message to tell you that the file does not exist. If you just press RETURN, *cedra* assumes you are creating a file. In this case an empty grid display appears.

When the cursor is in the grid area you can edit the picture pixel by pixel, use commands to draw in the grid or merge the picture with another.

5.3.1.6 *Work area*

The work area maintains a display of the picture being edited to show you what it would look like on the normal PERQ screen.

5.3.1.7 *Picture areas*

When *cedra* is invoked, the area at the bottom of the screen is divided into 40 squares, each of which is large enough to hold an actual cursor.

The cursor you are editing is not necessarily shown in a cursor area. The cursor areas are available for you to temporarily store cursors for reference or for merging with the cursor being edited. It is good practice to store versions of the cursor you are editing to prevent drastic mistakes wiping out all the work. Cursors can only be written to a file from a cursor area.

5.3.2 Using cedra

5.3.2.1 *Invoking cedra*

Cedra requires the WMS to be running.

To invoke *cedra* just type:

cedra

in the selected window.

Cedra creates and selects a window the full size of the screen (excluding the WMS echo area). You are prompted to select this window so that *cedra* can receive keyboard and tablet input.

Initially *cedra* displays a prompt in the grid area asking you for the name of the cursor you want to edit.

If you do not give a filename but press **RETURN**, *cedra* assumes that you are going to create a cursor.

5.3.2.2 *Editing commands*

Initially the environment window displays the message:

No command

Moving the *cedra* cursor into the picture areas moves *cedra* into the Select environment. The buttonhole shows that you can *put*, get or *insert* cursors into this area. *Put* places the cursor being edited into the pointed at cursor area overwriting any cursor already there. *Get* displays a copy of the cursor being pointed at in the grid area. *Insert* places a copy of the cursor being edited in the pointed at picture area, moving any display already there into the adjacent picture area.

All commands to *cedra* are entered by selecting from a pop up menu. Moving the *cedra* cursor into any area other than the picture areas enables you to get a menu of commands. The mousehole shows the following alternatives; menu or exit. If the *cedra* cursor is in the grid area the environment changes from No Command to Edit. This environment provides a *Menu* button but also allows you to edit the grid area pixels by hand, setting them black or white. Pressing *menu* changes the environment to Menu and displays a menu of commands. Table 5.2 lists all these commands describing their use and effect.

If you give *cedra* a filename, *cedra* searches the current directory for the named cursor. If *cedra* finds the file the cursor is displayed in the grid area and the filename of the cursor is displayed in the filename window. The Work Area displays the curser in its true size.

The available menu commands may be divided into three categories:

1 General commands that control *cedra*

2 Element commands that act on the pixel representation of the cursor

3 Utility commands that provide utility type facilities within cedra

Each command corresponds to an environment. Once a command has been selected, further input may be required; from the keyboard in response to prompts in the message area, from the puck as indicated by the mousehole or by selection from a further menu of alternatives.

You can get rid of the menu without selecting a command by pressing the *Select* button outside the menu area.

The *Exit* option on the mousehole, exits the current environment and reverts to the original environment for that functional area.

Table 5.2
Cedra commands

Command	*Purpose*	*Use*	*Effects*
Read file	Informs *cedra* that you want to edit a cursor file	Prompt in Message area asks for the name of the file.	Changes environment to Read.
			Displays file in the grid area and in the picture area.
		Type the filename followed by RETURN.	The first in an animated file is displayed in the grid area and the whole file is read into successive picture areas.
		Just type RETURN to quit the environment	
Write one	Saves an edited cursor in a cursor file	Prompt in message area asks for the filename.	Changes environment to Write.
		Type the filename followed by RETURN or just RETURN to abort.	Whilst waiting for filename, *cedra* cursor disappears.
		Move *cedra* cursor to cursor area to point at the cursor to be written.	After filename given, *cedra* cursor becomes a writing pencil. Buttonhole shows *Select* or *Exit*.
		Press the *Select* button to write the file.	Writes selected cursor to named file.
Write all	Saves all the cursors in the cursor in one file as an animated cursor type file	Prompt in Message area asks for the filename.	Changes environment to Write.
		Type the filename followed by RETURN or just RETURN to abort	All cursors written to the named file
Help	Displays the name of the help file */usr/lib/cfedra/ /cefdra.hlp*		Changes environment to Help momentarily.
			You need to enter *spy* to look at this file First you need to.quit *cedra.* The help file is a list of *cedra* commands together with a brief description of each command.
Quit	Quits *cedra*	If cursors have been changed (a *put* has been done) and have not been written, prompt in message area asks:	Changes environment to Quit.
			If response to update prompt is No mousehole offers the opportunity to get a menu so you can write the file.
		Throw away?	
		Type Y or N. Default is N (no)	

Table 5.2 (continued)

Command	*Purpose*	*Use*	*Effects*
Set all	Inverts, clears or sets all pixels in the Grid area	Press required button	Changes environment to Set all. Mousehole offers *Invert*, *Black* or *white*
Join	Combines two cursor pictures	Select cursor to join with cursor in grid area by pointing the *cedra* cursor at a pattern in the picture areas and pressing any button. Abort by pressing any button outside the picture areas	In picture areas any button press selects the pointed at cursor and combines it (either logical OR or logical AND) with the cursor pattern in the grid area. *Join* assumes the origin of the new pattern to be the origin of the original cursor in the Grid area. A button press outside the cursor areas aborts the command
Move	Moves pattern in grid area right, left, up or down	Press button to select required direction of move	Changes environment to Move. Mousehole offers L/R, Up/Down or Exit. Selecting L/R changes mousehole to offer Left, Right or SELECT. Up/Down changes mousehole to offer Up, Down or Select. Select reoffers L/R, Up/Down or Exit. Button press moves the pattern one pixel in the required direction. Holding the button down causes successive moves.
Rotate	Rotates pattern round one of 3 dimensional axes	Press button to select required type of rotation	Changes environment to Rotate. Mousehole offers Rotate, Mirror or Exit. Selecting Rotate changes mousehole to offer +90, -90 or SELECT. Mirror changes mousehole to offer Roll, Spin or SELECT. Rotate is a move in the plane of the grid. *Spin* is a 180° turn round a vertical axis in the middle of the grid area (28th bit line).

Table 5.2 (continued)

Command	*Purpose*	*Use*	*Effects*
			Roll is a 180° turn round a horizontal axis in the middle of the Grid area (32nd bit line). Select reoffers L/R, Up/Down or Exit
Line	Paints a line in black, white or the inverse of the current state	Press button to select colour of line. Draw line by pressing and holding *Paint* button. First press signals beginning of line. Release signals end of line. You can draw as many lines as you like by successively pressing and releasing the *Paint* button All lines are drawn in the selected colour. Reselect the colour by pressing the *colour* button and then selecting one of the colours	Changes environment to line. Mousehole offers choices for the colour of the line; *black, white* or *invert.* Abort the command by moving cursor outside the grid area. Once you have selected a colour, the cursor cannot go outside the grid area while painting. After colour selected buttonhole offers *Paint, Colour* or *Exit* *Colour* reoffers the original colour selection. After colour selected buttonhole reoffers *Draw, Colour* or *Exit*
Rectangle	Paints a rectangle white, black or the inverse of the current state	Press button to select colour of rectangle. Press and hole down *Paint* button. Press fixes one corner. Release fixes the opposite corner. You can draw as many rectangles as you like by successively pressing, drawing and releasing the *Paint* button. All rectangles drawn in the selected colour. Reselect the colour by pressing *Colour* and then selecting one of the offered colours	Changes environment to Rectangle. Mousehole offers choices for the colour of the rectangle; *black, white* or *invert.* After colour selected buttonhole offers *Paint, Colour* or *Exit.* *Colour* changes buttonhole to reoffer the colour choices. After colour selected buttonhole reoffers *Paint, Colour, Exit.*

Table 5.2 (continued)

Command	*Purpose*	*Use*	*Effects*
Display	Changes *cedra* cursor to the work area cursor to display its appearance in use	Select the cursor function from the menu offered. Use puck to move displayed cursor	Changes environment to Display. Menu of cursor functions displayed: CFNormal CFInvert CFCursComp CFInvCurComp CFWhitehole CRFBlackhole. Abort by selecting outside the menu. Work area cursor displayed with the selected function in place of the *cedra* cursor, following the puck. Mousehole offers *Exit* on all buttons
Animate	Displays an animated cursor in the Work Area. Assumes that all the cursors in the picture areas are steps in an animated cursor file and cycles through them	Select *Step* or *Cycle* Select direction of cycle.	Changes environment to Animate. Offers *Step, Cycle* or *Exit.* *Cycle* displays all the cursors in quick succession, that is, an animate display. *Step* displays the next cursor in the collection. Each button press displays one cursor the next in the cycle Mousehole offers, *left, right* or *Exit.* On exit, the last cursor displayed in the work area becomes the cursor displayed in the grid area
Set origin	Sets the origin of the cursor in the Grid area. Origin can be outside Grid Area. Default origin is (0,0), that is, the top left hand corner of the Grid area. Coordinates range from 0-(64K-1). It is good practice to set the origin first and draw from it	Type X coordinate of origin (0-(64K-1)) followed by RETURN or just RETURN to accept default. Type Y coordinate of origin (0-(64K-1)) followed by RETURN or just RETURN to accept the default	Changes environment to set origin. Prompt in Message area asks for X coordinate with the current value (the default) in brackets. After X coordinate given Message area prompts for Y coordinate with the current value (the default) in brackets

Table 5.2 (continued)

Command	*Purpose*	*Use*	*Effects*
Delete	Delete cursor from picture areas	Press *delete* to delete the cursor pointed at in the picture area You can delete as many cursors as you like before pressing *exit*	Changes environment to Delete. Outside picture areas mousehole offers *Exit*. Inside picture areas *cedra* cursor becomes a sword. Buttonhole offers *delete* or *exit*
Refresh	Restore screen from core table	Only use refresh if you are unsure of the screen content.	The result should not be different from the previous state of the screen.

6.1 Introduction

Under PNX, I/O devices are hidden and appear to be part of the file system. The PNX file system consists of four types of files; ordinary files containing text or an executable program, directory files which hold the names of files or other directories, special files which correspond to input or output devices and window description files. Window descriptions are not special files but when open behave comparably. By convention, special files are held in the directory */dev*. Special files provide the same sort of interface as ordinary files. Section 4 of the UPM describes the characteristics of each special file that refers to an I/O device.

A device is introduced to the PNX file system using */etc/mknod*. */etc/mknod* needs four parameters to define the device. The first parameter is the name of the file to represent the device. The filename is conventionally placed in the directory */dev*.

Secondly, */etc/mknod* needs to know the device type. Devices can be one of two types; structured or unstructured. Historically, the different types were labelled block structured or character structured. The second parameter has the value **b** for block structured (unstructured) devices or **c** for character structured (structured) devices.

A third parameter, the major device number, identifies the device driver to be used with the device. In fact, the major device number is an index to a system table containing pointers to device dependent routines. There are different system tables for structured and unstructured devices.

A fourth parameter, the minor device number, is used by the device dependent routines to modify their action. Typically, the minor device number selects a sub-device.

Example

The declaration:

etc/mknod /dev/cg c 6 36

attaches the ICL 6202/02 Correspondence printer as an output device.

The **c** parameter identifies the device as unstructured. The major device number, **6**, identifies the GPIB interface. The minor device number, **36**, identifies the device as a printer and contains a component to specify the GPIB address.

Device dependent routines that perform the I/O make up the device driver. Each device driver provided under PNX provides the following routines; *open, close, read, write* and special *ioctl*. PNX provides device drivers for:

1 Floppy disc

2 Fixed disc

3 Keyboard and display (including screen and window drivers)

4 Tablet and puck

5 Error device (for kernel error reporting)

6 Speech hardware

7 RS232C:

   (a) the interfaces themselves

   (b) the ICL 3185 Matrix Printer connected to an RS232C interface

   (c) the ICL 6202/03 Correspondence Printer connected to an RS232C interface

(d) a teletype control terminal remotely connected to an RS232C interface

(e) synchronous communications (2780/3780). See Chapter 7

8 GPIB

(a) the interface itself

(b) the ICL 6202/02 Correspondence Printer connected to the GPIB

(c) the ICL 6203 Electrostatic Printer/plotter connected to the GPIB

(d) the optional Bitpad tablet

Table 6.1 lists all the devices that have been declared to PNX using */etc/mknod.*

Table 6.1
PNX special files

Device	*type*	*major*	*minor*	*use*
/dev/flop	b	1	1	To access standard PNX floppy (also equivalent to old device) dev/rawflop
/dev/bflop	b	1	0	Standard PNX bootfloppy
/dev/formatfd	b	1	4	To format PNX floppy
/dev/fdplx	b	1	5	RT-11 type floppy
/dev/oldflop	b	1	6	Old PNX floppy
/dev/oldbflop	b	1	7	Old PNX boot floppy
/dev/hard	b	0	0	Fixed disc
/dev/veryhard	b	0	1	
/dev/tablet	c	12	0	Standard tablet, absolute tracking
	c	12	1	Standard tablet, relative tracking
	c	6	136	Optional gpib tablet, absolute
	c	6	200	gpib tablet, relative
/dev/*name*	c	6	31	To talk to gpib bus
/dev/*name*	c	6	32	To talk to gpib TMS 9914 regs
/dev/err	c	3	0	error device
/dev/rs	c	4	0	To drive RS232C (PERQ1)
/dev/rsA	c	4	0	To drive RS232C port A
/dev/rsB	c	5	0	To drive RS232C port B
/dev/rstty	c	4 or 5	3	To drive remote terminal on RS232C port allowing *uucp*

Table 6.1 (continued)

Device	*type*	*major*	*minor*	*Use*
/dev/lp	c	4 or 5	1	Matrix printer
/dev/cg	c	6	36	Correspondence printer on gpib
/dev/cr	c	4 or 5	2	Correspondence printer on RS232C
/dev/vp	c	6	35	Electrostatic Printer/Plotter
/dev/console	c	0	0	
dev/screen	c	0	8	Graphic output to screen
/dev/speech	c	7	0	Speech driver
/dev/tty	c	1	0	Control terminal
/dev/window	c	9	N(window ID)	Direct access to window

The following sections describe how to access the devices and their drivers. The printers are described separately from their interfaces as much of the printer support is device independent.

6.2 Floppy disc

flop (4) describes the special files which represent the floppy disc device supported under PNX. A floppy disc can be used in one of two ways:

1 AS AN RT-11 FLOPPY DISC This is useful for exchanging files between a PERQ running under PNX and:

   (a) a PERQ running under the PERQ Operating System (POS)

   (b) any other machine that supports RT-11 floppy discs

2 AS PART OF THE PNX FILE SYSTEM This enables you to:

   (a) exchange files between PERQ users

   (b) hold security copies on the floppy disc of important files on the fixed disc

   (c) receive software from your supplier

There are three stages to using a new floppy disc:

1 Format it

2 Set up a directory on it

3 Access it

6.2.1 Formatting a floppy disc

Formatting is the process of writing headers on each sector of each track. This determines the order in which sectors are read or written.

To format a floppy disc, run the *fl*(1) utility and issue the **format** command.

The **format** command formats both single and double density discs. The default is double density so if you want the disc to be single density you must specify the **/singledensity** switch before issuing the **format** command.

Discs can also be single sided or double sided. Double sided is the default so if you want the disc to be single sided you must specify the **/singlesided** switch before issuing the format command.

A parameter to the format command enables you to specify an interleave pattern to determine the order in which sectors are written and accessed. You specify the pattern by listing the order in which the sectors on a track are to be written. A default interleave pattern is provided for both double and single density discs as follows:

1 10 19 2 11 20 3 12 21 4 13 22 5 14 23 6 15 24 7 16 25 8 17 26 9 18

Boot floppies should be formatted using the *bformat* command. Boot floppies are single density. The interleave pattern for boot floppies is:

1 14 8 21 2 15 9 22 3 16 10 23 4 17 11 24 5 18 12 25 6 19 13 26 7 20

Formatting can not only interleave sectors but can also skew tracks. Skew is the offset between the first sector in each track. For example, if the skew is eight, the first sector read in the first track is the first physical sector but the first sector read in the second track is the ninth physical sector. The **skew** command must be specified before the **format** command. The default skew is 0, that is, no skew at all. Skew is available for you to try to gain performance improvements but for most uses any improvement is likely to be marginal.

The format command accesses the device */dev/format* (see *flop*(4)).

6.2.2 Setting up a directory

You can set up two types of directory:

1 An RT-11 directory

2 A PNX file system directory

To set up an RT-11 directory, run the *fl*(1) utility and issue the **zero** command.

To set up a file system floppy, issue the *mkflop*(1) command.

You can set up an empty directory on a used floppy. This clears the old directory, making all the old files inaccessible. It is, therefore, possible to change a file system floppy to an RT-11 floppy or the other way around.

6.2.3 Accessing a floppy disc

An RT-11 floppy is read and written by running *fl*(1) and issuing the **get, sget** and **put** commands. These commands use the device */dev/fdplx*.

A PNX file system must first be attached to the system using *mountflop*(1) . It can then be accessed using any of the normal system calls. The *open* system call determines the density. Double density floppy discs can only be accessed by */dev/flop*. Any attempt to *open* a file on a double density disc other than by */dev/flop* fails. */dev/flop* is equivalent to the old device */dev/raw/flop* which read sectors in physical order. */dev/flop* reads sectors in logical order determined by the hardware interleaving of the *fl*(1) **format** command.

6.3 Fixed disc

The fixed disc is not usually accessed directly. Ordinary files and directories on the fixed disc are accessed using system calls issued by utilities or programs.

It is possible to access the fixed disc as a single device using one of the special files */dev/hard* or */dev/veryhard* (see *hard*(4)).

6.4 Keyboard and display

While WMS is running, both the keyboard and display are controlled via Window Manager and are not usually accessed by any special file.

When a window is selected, all keyboard input is directed to the window and the selected window and keyboard together make a virtual terminal. Keyboard input is interpreted as described in *tty*(4). Keyboard characteristics may be set using *stty*(1) and may be changed using *ioctl*(2) calls (see *tty*(4)). All keys are programmable. Appendix 6 lists the raw values generated by each key press and shows the translated ASCII value for each key press. The device */dev/tty* gives direct access to the virtual terminal of window and keyboard for processes running in that window.

Input is read from the keyboard by *read*(2) calls. For output to the display, use *write*(2) for text and *wdbyte*(2), *wline*(2) and *wrasop*(2) calls for graphics.

To output to a particular window on the display rather than the controlling window of a process, use the device */dev/windowN* where N is the window ID (see *window*(4)).

While WMS is not running, the devices */dev/console* and */dev/screen* give direct access to the PERQ VDU. */dev/console* acts in the same way as */dev/tty* and may be used for text output. */dev/screen* is designed for direct graphics output but may be used for text as it uses the */dev/console* driver (see *screen*(4)).

It is, however, strongly recommended that the WMS is used for all graphic output rather than */dev/screen*. In this release of PNX updating on screen images is much more efficient than in earlier releases so there is little performance gain in using */dev/screen*.

The PERQ function keys, labelled PF1, PF2, PF3 and PF4 on the keyboard may be, set to any string value using *setpf*(1). Once set to a particular string, pressing that function key has the identical effect to actually typing the string.

In addition to the normal typewriter keyboard, the PERQ keyboard has a number pad which includes the function keys described above, a set of number keys, and a set of punctuation keys.

6.5 Tablets

The standard PERQ tablet connects to its own interface on the PERQ processor box. The optional high resolution tablet connects to the PERQ on the GPIB interface.

PNX determines the position of the puck on the tablet from tablet X, Y coordinates which it translates into screen coordinates to determine where the cursor is pointing. The puck position on the tablet can be absolute, that is, the X, Y coordinates specifying an actual position, or relative, that is, the change in X, Y coordinates specifying the line or movement from an initial position. The actual mapping structure is defined in *tablet*(4).

The special file */dev/tablet* controls the tablet. */dev/tablet* is declared to PNX as follows:

/etc/mknod/dev/tablet c *maj min*
where:

maj=12, min=0	specifies standard tablet, absolute tracking
maj=12, min=1	specifies standard tablet, relative tracking
maj= 6, min=>127	specifies the high resolution tablet on the GPIB interface. This is an eight bit number, the lower 5 bits signal the GPIB address (usually 8), the seventh bit if on signals relative tracking and the eighth bit signals that the device is a tablet

An *open*(2) call to */dev/tablet* switches on the tablet. A *close*(2) call to */dev/tablet* turns off the tablet as long as no other processes have the tablet open. These are the only systems calls that may be used with */dev/tablet*.

Window Manager keeps */dev/tablet* open but if using */dev/screen* you need to keep */dev/tablet* open for *wgread*(2) to work. Once enabled, *wgread*(2) calls to a window or to */dev/screen* return a value specifying the position of the puck on the tablet and bit values to describe which of the logical puck buttons are being pressed.

On the standard tablet:

Button 0 = the left button
Button 1 = the middle button
Button 2 = the right button

On the optional tablet:

Button 0 = the yellow button
Button 1 = the white button
Button 2 = the blue button
Button 3 = the green button

Button presses are interpreted as logical button presses. Logical buttons A, B and C are mapped on to physical buttons 0, 1, 2 by default and a press on button 3 (green) is interpreted as a press on logical button C. You can change the mappings using *wprofile*(5). In particular you may wish to set a value for logical button D in order to get four options from the four button puck.

On the standard tablet the default is:

A = left or yellow
B = middle or white
C = right or blue

C also = green for backward compatibility. Programs that use four buttons will still work but they cannot distinguish blue or green).

Programs written to interpret physical button presses need to take into account this logical to physical mapping.

6.6 Error device

If the kernel spots a hardware error and the kernel process wants to output an error message, the kernel process writes the message to */dev/err*. Any process may write a message to */dev/err*.

/dev/err/ creates an error record for each message received in a set format (see *errfile*(5)) and places the error record in a buffer.

A background process or daemon, *printerr*(1), continually reads the */dev/err* buffer. When it encounters an error record it displays the appropriate message in an error window, opening the error window description file if necessary. The error window may be deleted at any time in the same way as any other WMS window.

If WMS is not running, *printerr* prints the message to the screen.

6.7 Speech handler

The PERQ produces sound by sending data to a Zilog SIO chip on the I/O board. This chip is connected to the speech hardware.

PNX provides a general purpose speech handler offering application level programs the facility to use the speech hardware.

The speech hardware must first be created as a special file using:

/etc/mknod */dev/speech* **c 7 0**

/dev/speech identifies the speech driver (see *speech*(4)).

/dev/speech may be driven by the system calls *open, close, write* and *ioctl*. Reads to */dev/speech* have no effect. A *write* sends a stream of bits to the hardware which interprets the stream to produce a noise. Special *ioctl* calls may control the SIO chip but these should only be used by those familiar with the speech hardware.

Regular bit patterns, that is, equal numbers of 0s and 1s repeated, for example, 1111000011110000 produce regular tones. In general, the higher the number of 1s in the regular pattern, the higher the tone.

Errors detected by the speech driver routines are reported using the standard UNIX error reporting mechnism. *errno* is set to the appropriate error condition and the system call returns a value of -1.

The following example program creates tones according to the given regular patterns. The number requested by the program is the number of 1's in the generated pattern. This is followed by an equal number of 0's and the whole bit pattern repeated to fill a buffer. The buffer is then writen to the hardware to produce a tone.

Example

```
extern int errno;

main()
{
        register int i;
        register int ptrnlen;
        register int count;
        register char *bp;

        char buf[2000];
        int fd;
        int temp;

        fd = open( "/dev/speech", 2 );
        printf( "errno is %x\n", errno );

        while
        ( printf( "enter the length of 1's and 0's" )
        , scanf( "%d", &temp)
        , ptrnlen = temp ) {

        count = 0;
        for ( bp = buf; bp < buf + sizeof(buf); ++bp )
          for ( i = 0; i < 8; ++i )
            if (++count <= ptrnlen)
                *bp != (0x80 >> i);
            else {
              if (count == 2*ptrnlen) count = 0;
              *bp &= ~(0x80 >> i);
              }

        write( fd, buf, sizeof( buf ) );
        }
}
```

The following example shows a program which outputs speech recorded on a speech input device. Given suitable speech input hardware, that is, a device that records speech in the same way that the SIO chip outputs tones, the PERQ can talk or sing using this program.

Example

```
#include <sgtty.h>
#include "/usr/include/sys/speech.h"

main()
{
   union speechb S;
   char buff[512];
   int fd;
   int count;

   fd = open ("/dev/speech",1); /* open for write */

   if (ioctl (fd, TIOCGETP, &S) = = -1)
            { print f ("TIOCGETP failed");
              exit (-1);
            }
S.sp__regs.sp__XmitRate = 9;
if ioctl (fd, TIOCSETP, &S) = = -1)
            { printf ("TIOCSETP failed");
              exit (-1);
            }

while (count = read (0, buff, 512)
            {write (fd, buff, count);
            }
}
```

The default rate for speech output is approximately 3,000 bytes/second but *ioctl* calls allows you to set the transmit or write rate to the recorded rate. Normal speed is approximately 31,000 bytes in 17 seconds.

6.8 RS232C interface

The PERQ RS232C interface is described in the publication *ICL PERQ: User Guide.* The RS232C driver provides application programs with the facility to drive any single device or communications line connected to an RS232C interface within the capabilities of the PERQ I/O microprocessor. Before a device may be driven by the general purpose RS232 driver it must be created as a special file using the command:

/etc/mknod *filename* **c** *maj min*

filename is the name of the file to identify the device, conventionally */dev/rs* (see *rs*(4)). Devices on the RS232 A port must be declared with a major device number 4 and a minor device number 0. Devices on the RS232 B port must be declared with a major device number 5 and a minor device number 0. For PERQs with one RS232 interface, the RS232 interface is the same as for the RS232 A port.

Devices may then be accessed and driven using the standard PNX I/O system calls. *ioctl*(2) system calls control the RS232 interface. You can directly control the interface using the utility *srs*(1) which then issues *ioctl* system calls. The RS232 driver transfers data transparently.

Note: When using the RS232 driver in a multiprocessing environment, input data may be lost if the program loses control and the system buffer overflows.

6.8.1 Teletype driver

PNX provides facilities for driving an RS232 link to a terminal, or another computer, using the currently available terminal handling procedures (*tty*(4)). Additionally, PNX provides the facility to use a teletype terminal on the RS232 interface as a PNX control terminal. This facility is required if you want to use *uucp*(1) to receive communications from other UNIX systems. This facility may be included in the PNX initialisation procedure or may be invoked while the system is running.

Before the teletype driver may be used, the remote machine must be connected to an RS232 interface and created as a special file using the command:

/etc/mknod */dev/rstty* **c** *maj* **3**

/dev/rstty is the special file identifying the teletype terminal (see *rstty* (4)). The major device number should be 4 if the terminal is connected to RS232 port A or 5 if the terminal is connected to RS232 port B. For PERQs with one RS232 port, the port is the same as RS232 port A.

The teletype may then be accessed and driven using the standard PNX I/O system calls.

The file may also be opened by *init*(8) or by *rstty*(1) in which case the file may be used as a standard PNX control terminal.

If the file is to be opened at system initialisation by *init*(8), then the file *ttys*(5) must contain an entry for */dev/rstty.*

If the system is running, the utility *rstty*(1) initiates a teletype on the RS232 port for use as a control terminal.

6.9 GPIB interface

The PERQ GPIB interface is described in the publication *ICL PERQ: User Guide.* Each GPIB plug has a socket on its back so that you can connect more than one device to the GPIB.

To access a device on the GPIB you must first create a special file for that device using *mknod*(1), for example:

/etc/mknod */dev/filename* **c 6** *min*

where *filename* is the name of the device (conventionally the special file is placed in the directory */dev*). *Min* is the device address on the GPIB and can be from 0 to 30. If a high resolution tablet is used on the GPIB it is usually connected on address 8.

A special file set with minor device number 32 can be used to write to the TMS 9914 registers or read the sense data after an attention interrupt when exclusive control is in force.

A device on the GPIB is accessed using the system calls *open*(2), *close*(2), *read*(2), *write*(2) and *ioctl*(2).

6.9.1 Open

open(2) returns a file descriptor to identify the file for subsequent system calls. No input or output is initiated.

6.9.2 Read

read(2) reads data from the GPIB. The program can specify (using *ioctl*(2)) the following criteria for termination:

1 Buffer full

2 EOI received

3 Specify terminating character

You can also specify a time-out.

In all cases, the actual number of bytes transferred is returned as a result of the read.

The precise action of the GPIB driver depends on the minor device number. If the minor device number is 31, (the GPIB itself) then the PERQ waits for data and terminates according to parameters set under *ioctl*(2). If the minor device number is 0 to 30 and if 'no configure' (*ioctl*(2)) is not in force then all devices are untalked and unlistened. The PERQ is then set as a listener and the device as a talker. The read terminates according to parameters set by *ioctl*(2).

6.9.3 Write

write(2) transfers data onto the GPIB. The precise action of the GPIB driver depends on the minor device number and can be modified by *ioctl*(2). If the minor device number is 31, the *write*(2) call writes commands or data to the GPIB. If the minor device number is to 0 to 30, the *write*(2) call can also set all devices to untalk and unlisten, set the device as the listener and set the PERQ as the talker before writing the data.

6.9.4 Close

close(2) closes the device. No input or output is associated with the call though any outstanding transfer is aborted.

6.9.5 Ioctl

ioctl(2) codes applicable to GPIB devices are listed in *gpib*(4). These codes can determine termination characters and set the transfer mode, for example, high volume data transfers are possible. Also, a device can have exclusive control of the GPIB.

Some devices can generate service requests. An *ioctl* code signals that this is so and the device is then added to the queue if devices to be serially polled if an SRQ interrupt is received. A signal SIGGPIB *(signal*(2)) is sent to every program with a file open for a device indicating SRQ.

6.10 Printers

PNX supports the following printers:

ICL 3185 Matrix Printer
ICL 6202/02 Correspondence Printer
ICL 6202/03 Correspondence Printer
ICL 6203 Electrostatic Printer/Plotter

This section describes the software support for the above printers. Specific printer publications (see Preface) describe their hardware and operation.

Each of these printers is identified to PNX as a special file by *mknod*(1). The printer special files can be accessed directly using the standard PNX I/O system calls. The *open* call to a printer special file also initialises the printer to a particular default mode.

Standard UNIX provides *pr*(1), a utility to format files and print them on the standard output and *lpr*(1) a spooler to queue files for printing to a line printer. PNX provides extensions to these standard UNIX utilities, providing more spoolers for the different types of printing and providing a *plot*(1) utility to write the contents of the screen or a window to a file which can then be queued by a spooler for despatch to a printer.

There are three spoolers, one for each mode of printing; ordinary text printing, correspondence quality printing and plotting. Each printer offers one, two or all of these modes of printing. Despoolers take the file to be printed from the spooler queue and despatch it to the printer. There is a despooler for each mode of printing on each printer, six in all. Table 6.1 shows the relationship between the printers, the facilities they offer, the spoolers and the despoolers . Despoolers contain the device dependent code.

Table 6.2
PNX Printer Facilities

Printer	*Facilities*	*Spooler*	*Despooler*
ICL 3185 Matrix Printer	Ordinary printing characters made by a matrix of dots	lpr	lpdml
	Plotting. All points addressable mode	lpp	lpdmp
ICL 6202/02 Correspondence Printer	Correspondence quality printing. Characters made by impact as by a typewriter	lpc	lpdcg
ICL 6202/02 Correspondence Printer	As for the ICL 6202/02	lpc	lpdcr
ICL 6203 Electrostatic Printer	Print. Characters made up of dots	lpr	lpdel
	Plotting. Resolution 200 x 200 bpi	lpp	lpdep

6.10.1 Printer special files

The printers should be declared to */etc/mknod* as follows:

ICL 3185 Matrix Printer	**/etc/mknod**	**/dev/lp**	**c**	**(4 or 5)**	**1**
ICL 6202/02 Correspondence Printer	**/etc/mknod**	**/dev/cg**	**c**	**6**	**36**
ICL 6202/03 Correspondence Printer	**/etc/mknod**	**/dev/cr**	**c**	**(4 or 5)**	**1**
ICL 6203 Electrostatic Printer/Plotter	**/etc/mknod**	**/dev/vp**	**c**	**6**	**35**

Printers on the RS232 A port should be declared with a major device number 4 and printers on the RS232 B port should be declared with a major device number 5. PERQ 1 only has 1 port. This is equivalent to RS232 A.

These special files can be accessed directly using standard system calls. The *open* system call initialises each printer to a default printing mode. All I/O functions are standard for the interface, either RS232C or GPIB. Any external switch settings are not disabled for this direct access and must be set to the default mode. The default mode for each printer is described in the relevant printer section.

As long as the data is suitable, standard PNX utilities such as *cat*(1) can use the default mode on all printers.

6.10.2 Spoolers

Files may be printed in one of three ways.

1 The output of listing programs such as *cat* can be directed to the printer file. For example:

cat *filename* > **/dev/lp**

print *filename* on the Matrix Printer. The picture must be set up on the right RS232 port as specified for */dev/lp* and must have the switch to the default mode

2 The file can be queued by a spooling program

PNX provides three spooling programs as follows:

lpr(1) for printing
lpc(1) for correspondence quality printing
lpp(1) for plotting

A file is queued for printing by one of three commands:

lpr *filename* for ordinary printing

lpc *filename* for correspondence printing

lpp *filename* for plotting

Spoolers use the default state of the printers so printer switches must be set to their default values

3 The output of *pr* can be directed to *lpr* or *lpc*. *pr*(1) prints a file with a header at the top of every page. For example:

pr *filename* | **lpc**

Plot(1) copies the contents of a given screen or window to a file in a format governed by *plot.h* (see *plot*(5)). The output must be directed to *lpp:*

plot *filename* | **lpp**

The *plot* command issued in a window does not print the title or border of the window. A program *splot* is available to plot a window from within a process (see *plot*(1)). *Plot*(1) uses two library routines also available to users; *lplota*(3) copies a window into memory, *lplotb* writes a file of format given by *plot.h* from an area of memory.

Each spooling program takes a file from its standard input and calls the library routine *lpq*(3) to queue the file on its associated directory. Each spooler has an associated directory as follows:

lpr(1) */usr/spool/lpd*
lpr(1) */usr/spool/lpdc*
lpp(1) */usr/spool/lpdp*

Once the file is queued the spooler *execl's* its associated despooler.

Only one despooler is associated with a spooler at a given time. The file *spoolgov*(5) has a text entry for each spooler showing the name of the despooler currently associated with the spooler, whether or not the spooler is available for use and whether or not files queued by the spooler are to be despatched to the printer.

For example:

Spool directory	Despooler	Active	Closed
/usr/spool/lpd	/usr/lib/lpdel	1	0
/usr/spool/lpdc	/usr/lib/lpder	1	0
/usr/spool/lpdp	/usr/lib/lpdmp	1	0

For the active and closed columns, 1 means true, 0 means false. Active means that the associated despooler is taking files off the queue and sending to the printer. Closed means that files cannot be sent even if the despooler is active. You must close a spooler before unplugging a printer or information may be lost.

The device driver associated with the spooler may be changed while the system is running simply by editing *spoolgov*(5).

Despoolers call a library routine, *lpdq*, to take a file from the spooler queue for printing. The library routine always selects the file least recently created by the queueing routine *lpq*. Both these library routines provide the user with the name of the file being queued or dequeued. Users can *open* the file and *read* or *write* to it as necessary.

6.10.3 Despoolers

There are six despoolers, one for each printer in each of its standard modes of working (see Table 6.2). At any time, only one despooler is associated with a particular spooler, taking files from the spooler's queue directory and writing them to its printer.

The despoolers interpret function codes embedded in text data. There are two sets of function codes, one for each print spooler. The function codes interpreted by each despooler are listed in the relevant printer section.

If an error is encountered in printing or plotting a file and the output cannot be guaranteed to be correct, the despooler adds the file concerned onto the end of the spool queue for reprinting and goes on to print the next file in the queue.

If a printer cannot print, the despooler generates an error message describing the fault.

6.10.4 ICL 3185 Matrix Printer

The ICL 3185 Matrix Printer is suitable for general purpose text output and program development output. The Points Addressable mode enables graphic output including window or screen dump.

An *open* call to the special file */dev/lp* initialises the ICL 3185 Matrix Printer as follows:

Form length rotary switch enabled
Normal printing mode at 10 characters per inch and 6 lines per inch
All optional modes off

The printer can be driven by user programs using the standard PNX I/O system calls.

PNX provides two despoolers which output files to the ICL 3185 Matrix printer using the PNX spooler system. The despoolers are associated with the spoolers as follows:

lpr	*lpdml*
lpp	*lpdmp*

Files may be printed by issuing the *lpr* command or by sending the output of *pr*(1) to *lpr*. Files containing window or screen dump may be plotted by sending the output of plot(1) to *lpp* or by invoking *splot* (see *plot*(1)).

The printing driver, *lpdml* obeys function codes as listed in Table 6.3.

When plotting, the driver, *lpdmp*, assumes that the file to be plotted is formatted according to the format in *plot.h*. This file is included in the *plot*(1) program.

Note: The Matrix printer plots with a resolution of 78 dots/inch in the horizontal axis and 72.4 dots/inch in the vertical axis. The resulting plot is an enlargement of the PERQ screen which has a resolution of 100 dots/inch in both axes. The resultant plot is also slightly distorted as the PERQ points are square whereas the Martix printer dots are longer in the vertical axis.

Table 6.3
ICL 3185 Matrix Printer Function Codes

Function	*Code*	*lpdml*
Line Feed	LF	LF
Carriage Return	CR	CR
Form Feed	FF	FF
Horizontal Tab	HT	HT
Vertical Tab	VT	VT
Backspace	BS	BS
Select	DC1	DC1
Deselect	DC3	DC3
Load Tab position	DC4	DC4
10 cpi	RS	RS
12 cpi	GS	GS
17 cpi	FS	FS
Wide characters	US	US
Shift out	S0	S0
Shift in	S1	S1
Clear buffer	CAN	CAN
Ordinary characters	ESC0	ESC0
NLQ characters	ESC1	ESC1
Downline loadable characters	ESC2	ESC2
Set Top of form	ESC5	ESC5
6 lpi	ESC6	ESC6
8 lpi	ESC8	ESC8
Number of lines per page	ESC F nn (nn=0.0 to 9.9)	ESC F nn
Length of line spacing per page - N/2 inches	ESC G nn (nn=0.0 to 9.9)	ESC G nn
Number of character to character spaces for dot expansion line	ESC N n (n is a binary no. max 11)	ESC N n
Start of graphic printing with half dots	ESC % 1 nn (n & n are numbers of graphic codes)	ESC % 1 nn

Table 6.3 (continued)

Function	*Code*	*lpddml*
Line spacing of 1/144 inches multiplied by n	ESC % 9 n (n < 127)	ESC % 9 n
Loads 1 character pattern into loadable CG	ESC % A C (C is a code in range ^20 to ^5F)	ESC % A C
Directly skips number of lines designated	ESC VT nn (nn=0.0 to 9.9)	ESC VT nn
Sets a tab in HT memory	ESC HT n CR (n=various)	ESC HT n CR
Bold print ON/ OFF	ESC 0 ESC 8	
Shadow print ON/ OFF	ESC W ESC B	
Underscore ON/ OFF	ESC_ ESC R	
Half LF	ESC U	
Negative half LF	ESC D	
Print no data	ESC 7	
Repeat print	ESC N<n>	

6.10.5 ICL 6202/02 and ICL 6202/03 Correspondence Printers

The Correspondence printers provide high quality print with a large set of optional facilities controlled by function codes in the data.

An *open* system call to the special file */dev/cg* initialises the ICL 6202/02 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

An *open* system call to the special file */dev/cr* initialises the ICL 6202/03 as follows:

External switches enabled
Normal print at 6 lpi, 10 cpi
9600 baud

The Correspondence printers can be driven directly through standard PNX I/O system calls.

Files can also be printed by issuing the *lpc* command or by directing the output from *pr*(1) to the *lpc* spooler. PNX provides two despoolers, one for each correspondence printer, to print files queued by *lpc*. *lpdcg* drives the ICL 6202/02 and *lpdcr* drives the ICL 6202/03. Both despoolers only support A4 paper. The ICL 6202/02 should be on GPIB address 4.

Both despoolers enable the external print settings and obey function codes as listed in Table 6.4

Table 6.4
Correspondence Printer Function Codes

Function		*Code*	*lpdcg*	*lpdcr*
Line Feed		LF	CR,LF	CR,LF
Carriage Return		CR	CR	CR
Form feed		FF ESC VT'G' ESC RS ENQ ESC RS HT CR CR	FF ESC RS BEL ESC FF ACK ESC RS HT ESC FF 'B'	FF
Horizontal tab		HT	Modulus 8 spaces	Modulus 8 spaces
Vertical tab		VT	VT	VF
Backspace		BS	BS	BS
Bold print	ON/ OFF	ESC 0 ESC &	ESC 0 ESC &	ESC 0 ESC &
Shadow print	ON/ OFF	ESC W ESC &	ESC 0 ESC &	ESC W ESC &
Proportional spacing	ON/ OFF	ESC P ESC Q	ESC P ESC Q	ESC P ESC Q
Autobidirectional mode	ON/ OFF	ESC / ESC \	ESC / ESC \	ESC / ESC \
Underscore	ON/ OFF	ESC_ ESC R	ESC_ ESC R	ESC_ ESC R
Backwards printing	ON/ OFF	ESC 6 ESC 5	ESC 6 ESC 5	ESC 6 ESC 5
Absolute HT		ESC HT <n>	ESC HT <n>	ESC HT <n>
Absolute VT		ESC VT <n>	ESC VT <n>	ESC VT <n>
Set Left margin		ESC 9	ESC 9	ESC 9
Set right margin		ESC 0	ESC 0	ESC 0
Negative LF		ESC LF	ESC LF	ESC LF
Half LF		ESC U	ESC U	ESC U
Negative half LF		ESC D	ESC D	ESC D
Set VMI		ESC RS <n>	ESC RS <n>	ESC RS <n>
Set HMI		ESC US <n>	ESC US <n>	ESC US <n>
Set lines per page		ESC FF <n> ESC FF <N1,N2>	ESC F <n1,n2>	ESC F <n1,n2>
Ribbon Lift	UP DOWN	ESC B ESC A		ESC B ESC A

Table 6.4 (continued)

Function		*Code*	*lpdcg*	*lpdcr*
Reset		ESC SUB I ESC SUB R ESC CR P	ESC SUB I ESC SUB R ESC CR P	ESC SUB I ESC SUB R ESC CR P
Set HT stop at current carriage position		ESC 1	ESC1	
Clear all H and V tab stops		ESC 2	ESC 2	
Clear individual HT stop		ESC 8	ESC 8	
Clear VT stop at current position		ESC -	ESC -	
Set lower page margin		ESC L	ESC L	
Set top margin		ESC T	ESC T	
Clear top and bottom margins		ESC C	ESC C	
Graphics mode	ON/ OFF	ESC 3 ESC 4	ESC 3 ESC 4	Suppresses printing
Print	RED BLACK	ESC G ESC H	ESC A ESC B	
Print suppression		ESC 7	ESC 7	No data
Primary font		ESC I	ESC I	S0
		S 0	S 0	
Secondary font		ESC J S I	ESC J S I	S I
Proportional spacing justification		ESC M	ESC M	
Repeat print		ESC N <n>	ESC N <n>	repeat
End of data block marker		ESC E	ESC E	
Set HMI by pitch switch		ESC S	ESC S	

6.10.6 ICL 6203 Electrostatic Printer/Plotter

The ICL 6203 Electrostatic Printer/Plotter is designed for use as a general purpose medium speed text printer and high quality graphics printer. The graphics capability includes a screen or window dump facility. The printer must be connected on GPIB address 1.

An *open* system call to the special file */dev/vp* initialises the Electrostatic Printer/Plotter as follows:

PRINT mode
132 character per line
64 lines per page

The printer can be driven directly in this mode by standard PNX I/O system calls.

Text files may be printed by directing the output from *pr*(1) to the spooler *lpr*. PNX provides a driver, *lpdel*, which despatches files queued by *lpr* to the ICL 6203 Electrostatic Printer/Plotter. This driver obeys the function codes listed in Table 6.5.

Table 6.5
ICL 6203 Function Codes

Function	*Code*	*lpdel*
Line feed	LF	LF
Carriage Return	CR	CR
Form feed	FF	FF
Horizontal tab	HT	Modulus 8 spaces
Vertical tab	VT ESC VT nn (nn = 00 to 99)	VF (nn) line feeds

Screen or window dump is possible from program, user command or Window Manager menu invocation of *plot*(1). The output of *plot* may be stored as a file or sent to *lpp*. *splot* (see *plot*(1)) automatically sends to output of *plot*(1) to *lpp* for output to a printer.

The Electrostatic Printer/Plotter uses special electrostatic paper.

PERQ provides two RS232C ports (one on PERQ 1) and one Open Systems LAN port for connection to remote machines. RS232C connections to in-house, leased or dialled lines can support:

1 Direct transparent transfer. Useful for PNX to PNX transfer

2 *chatter*(1) A utility to emulate a teletype terminal

3 *rstty*(1) and *uucp*(1) *rstty*(1) initialises a remote machine, connected as device *rstty*(4), so that it can act as a PNX control terminal. While *rstty*(1) is running, PNX supports *uucp*(1), a utility to transfer files between PNX machines and other UNIX machines

4 2780/3780 emulation The 2780/3780 emulation package enables the PERQ to connect to any machines which normally connect to IBM 2780/3780 type terminals or to connect to any machine which emulates these terminals.

The 2780/3780 emulation package is a separate product which may be bought in addition to PNX. Using the emulation package is fully described in the publication *ICL PERQ:2780/3780 Communications under PNX*. Note: The 2780/3780 program only works over the RS232 A port on PERQ 2.

The Open Systems LAN connection provides two facilities

1 A transport service conforming to the ECMA-72 standard

2 A file transfer program *mftp*(1) which uses the Open Systems LAN transport service to transfer files between PERQs connected on an Open Systems LAN

The publication *ICL PERQ:Establishing IPA under PNX* describes how to write programs to use the transport service.

chatter(1), *rstty*(4), *uucp*(1) and *mftp*(1) are described in detail in the specifications in Appendix 3 but the sections below describe simple use of these facilities.

7.1 Direct transfer

To directly transfer files between two PNX machines connected over an RS232 link, follow these steps:

1 Ensure both PERQs have the same *srs*(1) configuration

2 Send files with the command:

cat *filename* > /dev/rs

Receive files with the command

cat /dev/rs > *filename*

PERQ can connect to any device that supports an RS232C connection. The special files */dev/rs, /dev/rsa, /dev/rsb* (see *rs*(4)) use the general RS232 interface driver.

The RS232C driver may be used from within a program. To send or receive first *open*(2) the device then use *ioctl*(2) calls to set the parameters to define the RS232C characteristics. Send and receive using *write*(2) and *read*(2) calls.

Alternatively you can set the parameters from the keyboard using srs(1).

Direct transfer works for normal text files as they are unlikely to contain the termination character set by *srs*(1) (default is ^C). However, binary files cannot be transferred in this way as they are certain to contain the termination character in which case the transfer will stop as soon as this character is encountered.

Binary files may be transferred in **raw** mode (see *srs*(1)) but then there is the problem of signalling the end of the file.

7.2 Chatter(1)

Chatter(1) enables the PERQ to act as a teletype terminal for some remote mainframe connected to the PERQ via the RS232C interface. The PERQ must be registered as a terminal with the mainframe. The available facilities are precisely those of a normal teletype terminal for that mainframe. You can send command lines to the remote mainframe in response to prompts from the main frame and receive the result of commands, transmit files and save the mainframe output in a file.

The link between the PERQ and the mainframe is controlled by the mainframe. Local actions by the PERQ operator such as saving mainframe input to a disc file or transmitting a file to the mainframe do not affect the operation as a teletype. Received files are listed to the screen (or window). While *chatter*(1) is running, input from the keyboard does not have the normal PNX effect (see *tty*(4)). Key presses produce their raw key value and may or may not have an effect on the remote mainframe.

Chatter may be used to transfer files between two PERQs connected together. The facility is similar to *cat* but slower. However, any type of file may be transferred.

7.2.1 Using chatter(1)

1 Ensure that the physical link between the PERQ and the mainframe exists. The link is most likely to be a dial up connection with asynchronous modems at either end set at the speed advised by the mainframe site or by local ICL staff. This is the concern of the network manager (see section 2.3.4)

2 Invoke the program by typing:

chatter *device*

This establishes the logical link between the keyboard and screen (or selected window) and the remote mainframe. *device* may be *rs, rsA* or *rsB* The default is *rs* (identified with *rsa* on PERQ 2) and this is used if anything else or nothing is typed

3 Send input lines to the remote mainframe, for example:

inf(filename) /* for VME */

This input line is directed to the mainframe and echoed by the mainframe onto the PERQ screen. The input lines is executed if it is understood by the mainframe

4 Press ^K to get a menu of *chatter* commands (see *chatter*(1) for a list). This command switches off the transparent mode of the keyboard and expects one of the commands to be entered. After any input, either a valid command or anything else, the keyboard again becomes transparent. For every command you wish to enter you must first press **^K** to enter command mode

5 SENDING Use the **T** command to transfer a whole file transparently.

The *transmit* command is reasonable for PERQ to PERQ links with *chatter* on each end but for sending files to a real machine it is best to use the *input* command

Use the **I** command to send a file a line at a time in response to a mainframe prompt

Input simulates an operator inputting one line at a time in response to a prompt from the mainframe.

In *chatter* you specify the prompt string expected and then *chatter* sends the next line of the local file every time it receives this prompt.

Note: You may need to change the PERQ end of line character to signify the end of an input line (to ^C for VME). *Input* does not work if the mainframe expects CR as a record terminator.

Files may only be sent with **^C** or new-line as a record terminator. For mainframes that insist on CR you can enter input lines as follows. Issue the command:

stty -crmod

Thereafter use **LF** instead of **RETURN** to terminate a command:

chatter <lf>

^K

Save <lf>

To get back to normal issue the command:

stty crmod <lf>

Thereafter, use **RETURN** to terminate a command

RECEIVING Use the S command to set up a file to receive any mainframe output and send a command line to the mainframe requesting it to list the file

6 To leave CHATTER:

(a) Use the C command to close any file set up to receive from the mainframe

(b) Use the Q command to stop *chatter*(1) and return to the PNX shell

7.3 *rstty*(1) and *uucp*(1)

The special file */dev/rstty* is an interface to another terminal connected to the PERQ via an RS232C interface. This terminal interface enables the remote terminal to log in to the PERQ. This remote logging in is necessary to allow a remote UNIX machine to use *uucp*(1), a UNIX to UNIX file transfer program. *uucp*(1) can work over an RS232C link to another UNIX machine since *uucp* can log in to standard UNIX but for PNX to PNX connection or a remote uucp request to PNX the remote machine must be the device designated by */dev/rstty*.

/dev/rstty can refer to a terminal on either or both of the RS232C ports (see *rstty*(4)).

Before *uucp*(1) can run over */dev/rstty*, the file must be open. */dev/rstty* can be opened when the system is initialised by *init*(8). If */dev/rstty* is not opened at initialisation, use the utility *rstty*(1) to open the file and initialise the remote terminal.

uucp is a suite of programs to permit communication between UNIX systems (and PNX) over dial up or hardwired communication lines. The *uucp* system consists of the following programs:

uucp	Creates work file and gathers data files in the *uucp* spool directory (*usr/spool/uucp*) for the transmission of files
uux	Creates work files, creates execute files and gathers data files for the remote execution of UNIX or PNX commands
uucico	Executes the work files for data transmission
uuxqrt	Executes the execution files
uuname	Lists the names of all the system in the *uucp* network, that is, lists *L.sys*
uulog	Updates the *uucp* log file and reports on the status of *uucp* requests
uuclean	Removes old files from the spool directory

These programs (except *uuname*) are described in articles 35 and 36 in *Volume 2B* of the *UPM*. A users primary interface with the system is by issuing the commands *uucp*(1) or *uux*(1).

Each command puts work and data files into */usr/spool/uucp* and initiates *uucico*. *uucico* checks that a connection exists and logs in to the remote system with the log in name *uucp*. The log in name *uucp* looks like any other user and is constrained by all the local protection rules.

7.3.1 Using uucp(1)

1 Ensure there is a physical link between the two systems. Local hardwired connections should be linked using a 'null-modem' cable. Dial up connections require a modem between the R232C port and the telephone link. To initiate *uucp* connections the modem must be an 'originate' or 'auto-dialler' modem. To just receive *uucp* and not initiate calls an 'auto-answer' modem is required. Modems at either end of the link must be matched for speed

2 Ensure the remote device is identified as */dev/rstty* (see *rstty*(1)). Modems must match *rstty* for speed and terminal parameters should be matched. The command *rstty*(1) takes a parameter to determine the type of terminal */dev/rstty* represents (see *getty*(8) for the types of terminals)

3 Issue a file transfer request as follows:

uucp [*option*] *sourcefile destinationfile*

Sourcefile and *destinationfile* are file descriptions designating the file to be sent and the file to receive the transmission. Either file may be the file on the remote system.

The remote filename must have the form:

systemname!pathname

Where *systemname* is the name of the remote system.

The command *uuname* lists the *uucp* names of known systems. The-l option to this command returns the local name.

Pathnames can be a full pathname or a pathname preceded by *~user* where *user* specifies a user name on the remote system. In the latter case, the remote system precedes the given pathname with the designated user's home directory. If a full pathname is not supplied, the given pathname is preceded by the current directory on the remote system. Shell metacharacters are valid.

Options are described in *uucp*(1).

The actual transmission is handled by a background process.

7.4 mftp(1)

mftp(1) is a file transfer program to support PERQ to PERQ connection, either PNX to PNX or POS to PNX, over an Open Systems LAN.

mftp uses the Open Systems LAN transport service. This transport service is fully described in the publication *ICL PERQ:Establishing IPA under PNX* but you do not need to know anything about the transport service to use *mftp*.

Every service offered by each machine on an Open System LAN is identified in a transport address file. The transport address file exists on every machine in the network. Setting up the transport address file is a system management function (see section 2.3.4.2 and *mknet*(1))

In any connection, the initiating machine is the master and the responding machine is the slave. In PNX machines the *slaveft*(1) program runs in the background. The mftp sets up the PERQ as the master and establishes a connection with the *slaveft*(1) program running in the remote PNX machine.

A PNX machine cannot initiate a connection with a POS machine. A PERQ running POS is a single process machine so cannot run the slave program in the background. For a PNX machine to connect to a POS machine, the POS machine must be running the PERQFTP program. This probably necessitates an arrangement between the two operators so even though the POS machine must be the master in a PNX to POS connection this is no extra restriction on the facility other than the restriction imposed by POS as a single process system.

7.4.1 Using mftp

1 Issue the *mftp* command with the name of the machine you want to connect to:

mftp *systemname*

systemname is the name of the file in the transport address file that requests the *slaveft* service from the remote machine. Find out the filename from the system manager (see section 2.3.4.2) or by reading the transport address file (see *mknet*(1))

2 Login to the remote machine in the standard way. The username and password must be valid for the remote machine.

The login display shows the current remote pathname. This is the home directory of the login name

3 Issue commands to send or get files. You can change working directories in the local and remote machines, send files or get files and send files to a remote print or plot files. A list of valid commands is in *mftp*(1).

All files have their normal access permissions.

4 Quit MFTP with the quit command:

q

The programming languages available as an integral part of PNX are C and UNIX FORTRAN 77. Pascal is also available as a separate product.

8.1 Bibliography

8.1.1 C language bibliography

Information on developing C programs under PNX on the ICL PERQ is given in Appendix 5.

The definitive publication on the C language is:

The C Programming Language, Brian W Kernighan and Dennis M Ritchie, Prentice-Hall, 1978 which takes a tutorial approach to the subject, and includes a C reference manual as an appendix.

The ICL publication *UNIX Programmer's Manual* (*UPM*) contains several papers on C in Volume 2:

Section	*Title*	*Description*
14	The C programming Language - Reference Manual	A slightly modified copy of the appendix in *The C Programming Language*
15	Lint, a C Program Checker	Because of the flexibility of the language, C compilers cannot perform sophisticated checking *lint*(1) can, because it can risk being wrong
34	A Tour through the Portable C Compiler	How this compiler works

The publication *The Bell System Technical Journal* (July-August 1978, Volume 57, Number 6, Part 2) was a special edition concentrating on UNIX and associated matters. It contains the following articles of relevance to C:

Page	*Title*	*Description*
1991	The C Programming Language	A useful introductory outline of the language
2021	Portability of C Programs and the UNIX System	A discussion of the advantages of portable design, using C and UNIX as examples

8.1.2 UNIX FORTRAN 77 bibliography

Information on developing UNIX FORTRAN 77 programs under PNX on the ICL PERQ is given in Appendix 4.

The appropriate language reference documentation is the ICL publication:

FORTRAN 77 Language

8.1.3 Pascal bibliography

Information on developing Pascal programs under PNX on the ICL PERQ is given in:

ICL PERQ: Developing Pascal programs under PNX

The appropriate language reference documentation is the ICL publication:

ICL-Pascal Language

8.2 Mixed language programming

Programs written in C, FORTRAN 77 or Pascal may call procedures written in the other two languages and global data may be shared.

Details on mixing C and FORTRAN 77 are in Appendix 4. Details on mixing Pascal with C or FORTRAN 77 are in *ICL PERQ: Developing Pascal programs under PNX.*

8.3 Debugging

PNX provides *sdb*(1), the symbolic debugger, a utility for debugging C, Fortran 77 and mixed C and Fortran 77 programs. There is no debugging utility for Pascal programs.

Another aid to interpreting errors are error messages and exception codes. Most of these are self explanatory but the floating point exception codes are explained below (section 8.3.2)

8.3.1 *sdb*(1) the symbolic debugger

sdb provides two major debugging facilities:

1 After a program crash, *sdb* can use the core image file, a memory dump provided at the time of the crash, so that you can examine the state of the program at the time of the crash

2 Any program can be run under the control of *sdb* in various stages and the state of variables examined at points throughout the execution of the program.

If a program crashes, that is, terminates abnormally, a file is provided called the core image file. This core image is a snapshot reflecting the state of the program at the time of the crash. This image contains a dump of several areas of memory including the user area (information about the program, for example, the files and the current directory), kernel stack (system calls), user stack, static (global) data and the text area. *sdb* can access the reflection of the user area, kernel stack and user stack to provide information about the values of variables, functions called and the place where the crash occured.

The name for this core file is **core.** If you want to keep the core file for examination you should rename it using the *mv*(1) command.

sdb provides a set of commands for looking at source files. You can use *sdb* to find out where a crash occurred and then use these examination commands to examine the source file.

sdb also provides a set of commands to control the execution of a program. In *sdb* you can run the program and arrange for it to break at any point or you can run parts of the program separately. After any break there are commands to enable you to discover the value of variables. Other commands allow you to monitor the execution of the program until a particular variable's value changes.

8.3.1.1 *Invoking sdb*

The syntax is:

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

objectfile is an executable program file which must have been compiled with the **-g** option (debug)set. If you do not specify *objectfile* to *sdb* then *sdb* assumes that the file is **a.out**, the default output of the loader, *ld*(1).

Corefile is the core image file produced after a program crash. If you do not specify *corefile* them *sdb* assumes that the file is **core**, the default core image file. If **core** does not exist it does not matter.

Directory specifies the directory *sdb* is to search for the source files. *sdb* assumes that *objectfile* and *corefile* are in the current directory. If you do not specify *directory*, *sdb* assumes that source files are also in the current directory.

sdb searches the directory for all the files making up the program including the source files.

Whatever it is doing, *sdb* maintains the idea of a current line, a current procedure and a current file. If a core file exists then *sdb* initially sets the current line, current procedure and current file to the line, procedure and file containing the source statement at which the process stopped. If there is no core file, *sdb* initially sets the current line, current procedure and file to the first line in **main()**.

All lines in the program may be referred to by number. All *sdb* commands refer to the current line or to a particular line number. All the commands are listed in the *sdb*(1) specification.

The HELP COMMAND:

h

produces a display summarising all the *sdb* commands for ready reference.

Specifying lines

Line numbers are always relative to the beginning of the file containing the line. You can specify any line in a program either by specifying *filename:number* or by specifying *procedure:number*; in both cases number is relative to the beginning of the file.

If you do not specify a file for *filename* then *sdb* assumes the current file. If you do not specify *number, sdb* assumes the first line of the named file or procedure is intended.

Specifying variables

Names of variables should be written just as they appear in the program.

Variables local to a procedure may be accessed in the form:

procedure:variable

If no procedure name is supplied *sdb* assumes the current procedure.

Structure members can be specified by:

variable.member

Pointers to structure members can be specified by:

variable → member

Array elements may be specified by:

variable [*number*]

Combinations of these forms may be used to specify a particular variable.

Variables may also be accessed by their address. Address may be given in decimal, octal or hexadecimal. Addresses on the user stack (used when examining a crashed program) may not be specified in decimal.

8.3.1.2 *Examining a crashed program*

Probably the first thing you want to know is where the crash occurred. The command:

l

prints the last line executed.

You can also look at the stack to see the last variable that changed and its value or to see the order of changes. The command:

T

prints a trace of the top of the stack. The command:

t

prints a trace of the entire stack.

You can use the source file commands to move forward and backward through the source containing this last line or look at other source files making up the program (see *sdb*(1)).

You can use the data commands to print the value of any variable at the time of the crash, or the address of any variable or line in memory at the time of the crash (see *sdb*(1)).

You can examine any contiguous areas of memory to those returned by the above command, using the **new-line** command.

8.3.1.3 *Running a program under sdb*

There are four ways to run the program bit by bit:

1 Run the program until it reaches a specified breakpoint

2 Run the program in steps

3 Run the program until a specified variable change occurs

4 Run each procedure separately

Any one of these may be halted using the **k** command.

Specifying a breakpoint

Before running a program you need to set breakpoints wherever you want the program to stop. The command:

linenumber b *commands*

sets a breakpoint at a given *linenumber* (see 8.3.1.1 for details on specifying linenumbers). The program stops executing at the line before this line and any *commands* specified in commands are executed and then the program continues.

If no commands are specified the program just halts. Multiple commands may be specified, separated by semicolons. All *sdb* commands are valid so you can examine variable values or source files (see *sdb*(1)).

Breakpoints may be deleted by the **d** command or possibly ignored by commands with a *count* modifier. *Count* specifies the number of breakpoints to be ignored.

Running the program

The **r** or **R** commands run the program until a breakpoint is encountered. The *count* modifier specifies the number of breakpoints to be ignored.

After a break any commands associated with the breakpoint are executed and then the program continues. If no commands are associated with the break then the program halts.

You can input commands to examine data or some files or continue with the **c** or **C** command. This command allows you to specify an additional temporary breakpoint which is deleted after it is reached. You can continue from a point further on than the break using the **g** command.

Running the program in steps

The **s** or **S** commands run the program for a specified number of lines. **s** and **S** ignore any other breakpoints. **S** steps over subroutine calls.

Running the program until a change occurs

If you are interested in the behaviour of a particular variable the **m** and **M** commands allow you to run the program and halt wherever the value of a specified variable or the value at a particular address changes.

Running procedures

The *procedure* command executes the named procedure with any given arguments. One form of the command tells you if the procedure terminated successfully. An alternative form allows you to see the value returned by the procedure.

8.3.1.4 *Returning from sdb*

You do not have to quit *sdb* to execute any shell command. The ! command passes any named command to the shell for execution. After execution control is returned to *sdb*.

The **q** command quits *sdb* and returns to the shell.

8.3.2 Floating point exception codes

Most error messages are self explanatory but errors in floating point arithmetic produce the message:

floating point exception code n

where n is an integer; 0, 1 or 2. These integers have the following meaning:

0 Division by 0

1 General overflow:

(a) 64 bit overflow

(b) 32 bit overflow (16Kb WCS only; in 4Kb WCS, 32 bit real arithmetic is carried out to 64 bit range and precision until the result is to be stored when it is converted back to a 32 bit number)

(c) overflow converting 64 bit real to 32 bit real (64 bit reals have a much greater exponent range than 32 bit reals). The conversion may be explicit, in a register, or implicit when storing a 64 bit register to a 32 bit location, or when loading a 32 bit register from a 64 bit location

2 Real to integer conversion overflow

3 Illegal floating point instruction subcode. This can only occur if the program has been miscompiled or corrupted since being loaded

8.3.3 Error messages

Any user process can send messages such as error or status reports to */dev/err*. A daemon, *printerr*(1), prints the messages in the error window or on the screen if WMS is not running. See *err*(4) and *printerr*(1) and section 6.6 for further details.