Phototypesetter previewer initial specification

James Ashton

University of Wollongong

Follow this and additional works at: https://ro.uow.edu.au/compsciwp

Recommended Citation
https://ro.uow.edu.au/compsciwp/12

Research Online is the open access institutional repository for the University of Wollongong. For further information contact the UOW Library: research-pubs@uow.edu.au
Phototypesetter Previewer Initial Specification

by James Ashton

University of Wollongong

ABSTRACT

The current situation at Wollongong University regarding the phototypesetting of documents has two main inadequacies. Firstly the average time between the submission of a job and the receipt of a finished document is probably greater than a week. This is an unacceptably long time if one or more error correcting stages is required before an accurate document is produced. Secondly the cost of phototypesetting is high – currently sixty cents per (A4) page. The correction of errors can therefore be expensive as well as time consuming. A solution to these problems would be an easy method for accurately previewing output before it is submitted for setting. The graphics facilities required for such a previewer are provided by the I.C.L. Perq computer. This document gives the specifications of a phototypesetter previewer designed to run on the Perq.

November 4, 1984
Phototypesetter Previewer Initial Specification

by James Ashton

University of Wollongong

1. Introduction

High quality output is produced at the University of Wollongong's Computer Science department by a phototypesetter. This is driven by the troff (see TROFF(1)) word processing programme running under the Unix operating system. Because the phototypesetter is not connected to the department's Perkin Elmer, troff output is placed on a dummy printer queue which is transferred to the phototypesetter by magnetic tape periodically. For this reason there is usually a considerable delay in the production of documents.

The phototypesetter can produce an extremely wide range of character fonts and sizes. Characters are proportionally spaced both vertically and horizontally. Mathematical formulae and large tables can be produced. Because of the versatility of the output device, previewing is very difficult. The troff command line option -a gives an ascii approximation of output for a terminal or printer but this is useful only for text and gives at best a rough idea of layout. Using nroff (see (NROFF(1))) can be a useful aid but again the output is a bad approximation of the final result. To properly preview the phototypesetter, a device with good graphics facilities is required.

The I.C.L. Perq microcomputer recently acquired by the department has a high resolution bit mapped display of A4 size. This would be adequate to preview the phototypesetter to a high degree of accuracy - being able to show the whole of an average page of output with characters legible probably down to at least eight point size (characters down to five points can be produced but are rarely used).

2. Objectives

The objective of the project is to produce a system which allows the Perq to emulate the phototypesetter. The problems involved in this task dictate that a number of separate programmes be written. Firstly a set of simple routines must be written to take troff output in an appropriate format and transfer it (down a serial line) to the Perq. Next an interpreter to read these data and
produce the required displays is needed. Finally, a method of producing the diversity of characters required by the interpreter is required.

2.1. Utility Routines

Troff is capable of producing output in a large number of formats. The previewer must use a format which is neither too verbose or device dependent. Transfer of these data will be by the use of the cu (see CU(1)) command. It will be necessary to write some small routines to facilitate the easy generation of troff output in the correct format and the subsequent transmission of these data to the Perq.

2.2. The Interpreter

The job of the interpreter will be to read the troff output files and position characters of appropriate size and font in the right place accordingly. The programme will be interactive in nature and will allow the operator to view any page in the document. It will be able to deal with pages of varying lengths and widths though formats larger than A4 will not be able to be displayed fully at normal scale. There will be a facility to allow the reduction of pages so that large pages can be viewed all at once or to enlarge pages to facilitate closer examination of fine detail. If a suitable printer becomes available, it will be possible to obtain hard copy of the previewed output.

2.3. Character Generation

The large variety of character fonts and sizes makes it impossible to separately define each shape. Instead, the shape of each character in several fonts will be defined and an algorithm developed to shrink these down to the necessary size. Because the data used by the phototypesetter to produce characters is stored in an encrypted form, the character shapes must be manually digitised. A programme to do this using the Perq's graphics tablet device will be written.

The format for storage of the character shapes must allow for easy reduction. The two alternatives investigated were to store characters on a high resolution pixel map or as straight line segments connected to form polygons. The second alternative offers greater accuracy and ease of reduction as well as better data compaction. A polygon font editor will be written to allow the production of such characters and to store them in files readable by the troff interpreter.
3. System Development

3.1. Implementing a Simplified Previewer

The first phase of development will be to implement the first two programmes i.e. the utility routines and the interpreter. Without the character generation routines, the interpreter will be constrained to use the Perq’s default character set but the proportional spacing ability it has should make even this version useful. The interpreter’s more advanced features will be unimplemented at this stage and page previewing will be sequential only.

3.2. Adding Fonts and Random Access

Phase two will involve writing the polygon font editor and the character scaling algorithms. A single font (probably the TR or Times Regular) will be digitised at this stage to allow the testing of the scaling routine. A further advantage of using a digitised font over the standard font is that the characters will be designed to be proportionally spaced and special characters and ligatures will now print correctly. The troff interpreter will be improved to allow random access to pages.

3.3. Final Improvements and Debugging

Finally, in phase three, the interpreter will be brought to it’s final version with enlarging and reducing facilities. More fonts may be digitised if time allows.

4. Bibliography

The references to programming manuals refer to I.C.L.’s “Guide to P.N.X.” and Bell Labs’ “Unix Programmer’s Manual” both of which are supplied as documentation for the Perq computer.
Phototypesetter Previewer Design

by James Ashton

University of Wollongong

ABSTRACT

This document describes the design of a phototypesetter previewing system for the I.C.L. Perq microcomputer. It fully explains not only what functions the system will perform and how these facilities can be used, but also what basic methods are employed in their implementation. Some details concerning installation are also given.

November 4, 1984
1. Introduction

Because the phototypesetting programme is not implemented on the Perq, the documents must be prepared on the Computer Science Department's Perkin Elmer computers and transferred to the Perq via a serial line. This requires a number of small support programmes. The encrypted form of the fonts used by the phototypesetting device means that they are unavailable in a machine readable form and must therefore be manually digitised: The first of the two major programmes performs this task using the Perq's graphics tablet. The second major programme combines the document file with the font definition information to produce the previews in an interactive manner.

2. Overview

2.1. Document Production and Transfer

Documents for phototypesetting are prepared using a text editor in a format compatible with nroff and troff (see the "Nroff/Troff User's Manual"). To preview such documents, the following command should be executed.

\texttt{troff mcs -R -t document ^ bin > file}

The \texttt{-R} option to troff specifies that output will be in a relatively compact but device independent form. Normally, output (when using the mcs option) is piped through two filters but the \texttt{-R} (or raw) option prevents this as output from the first filter is too verbose for transfer via a slow serial line and output from the second is tailored for a particular device. Unfortunately raw output is of a binary nature, i.e. eight bits per character are used. The bin filter converts this data to an ascii format in a similar way od (see OD(1)) does, but more compactly and with the addition of checksum information. The cu (see CU(1)) command can then be used to transfer the resulting file to the Perq where it must be again filtered using unbin to restore it to a binary format. This is done with the command

\texttt{unbin < file > file2}

The resulting file may now be previewed.
2.2. Font Editing

The phototypesetter uses a number of fonts (currently eighteen) each of which has a two character ascii name and may contain up to two hundred and twenty four characters (octal 040 to 377). The first step in digitising is to phototypeset all the required characters as large as possible (currently seventy two point size) with a small marker such as a five point full stop at the lower left and one other corner of the enclosing box around each character. Thus if seventy two point characters are being used, the two points should be seventy two points (or one inch) apart independent of the width of each individual character. The points are for reference purposes. It may be helpful to further enlarge the characters using an enlarging photocopier or similar device as the larger the characters are, the easier they will be to digitise. The characters can now be affixed to the graphics tablet (taking care to ensure they are square with the tablet edges) for digitising. Because the tablet and multiple windows are in use, winit (see WINIT(1)) must be called before editing can begin. The digitising programme is edfont and should be called without argument. When the editing window appears on the screen and three small circles appear on it, the window should be selected using the puck's yellow button in the normal way. Messages will appear on the window from which the programme was called. Edfont will begin by editing character 040 octal in the TR or Times Regular font. The character and font may be changed interactively at any time. Each font is stored in a file with the same name as the font's name in a fixed directory. This directory may be changed during installation by changing a constant in the makefile file. New font files are created as required.

Font editing proceeds using both the four coloured keys on the graphics tablet's puck and the keyboard. The keyboard is used to enter single character commands, one per line, which control the mode of the puck's operation. In the mode to change characters or fonts, the new character and font are entered from the keyboard. In the different modes, each of the puck buttons retains a very similar function. The characters are defined as a number of polygons, thus the editor is concerned with generating and connecting points rather than in controlling each pixel in a grid. The characters seen on the screen are therefore shown in outline only. To digitise a character, it is first necessary to change to the appropriate font and character number. Then the puck must be scaled so that the position and size of the character fixed to the tablet map correctly on to the screen. This is done with the aid of the reference points discussed previously. Each time a new character is selected, the old one is saved. Each time a new font is selected, the font is compacted and replaced in a file with an appropriate name - fonts are held in a temporary file.
during editing. The new font is then copied to the temporary file. The compacting may take considerable time if the font has many characters defined in it.

2.3. Previewing

Like the font editor, the previewer can only be run after winit has been called. The previewer is called preview and requires the name of the document file as its only argument. Once called it will create a window covering almost all of the screen. This window should not be selected as commands are read from the window from which preview was called. The white key on the puck can be used to alternately view the display and calling windows as required. Commands are entered in a style similar to ed (see ED(1)). Decimal numbers with optional fractional parts are entered one per line. By default these request a preview of the page number given (numbered from zero) although by prefixing the number with various letters it is possible to change the page length or width, the scale at which previewing occurs and the horizontal position of the previewing. The letter "p" by itself will copy the current screen to a printer attached to the serial line. If a message indicating that "printfile" cannot be created appears, this indicates that a previous printer job has not yet finished. The command

The previewer knows the name of the directory in which the font definition files can be found. If this directory is changed during the installation of edfont, preview should be installed with a similar change made to its makefile file. If a font or character cannot be found, a definition from the TR font is substituted. The font files contain definitions in the form of polygons. There are a number of routines which convert these polygons to grids of pixels using scaling as required by the point size indicated in the document file. In this way a single font file can be used to represent the same font at a large range of sizes. While the output quality suffers, this is considered acceptable considering the storage savings made and the previewing nature of the output.

3. Design Details

3.1. Bin and Unbin

These two programmes are simple filters which read from standard input and write to standard output. They function identically except that unbin performs the inverse of bin and at the same time performs checksum verification. Bin takes a binary file and converts it to a file containing characters in the set "0" - "9", "A" - "F", "\n" and "X". Each line contains sixty four hexadecimal digits representing thirty two digits from the original file followed by
four hexadecimal digits, these being the checksum of the characters. The last line may contain 'X's to fill the space between the last character's representation and the final checksum. Both bin and unbin use buffering to speed their operation.

3.2. Edfont

3.2.1. Data Structures

As characters are defined as a number of polygons, a representation had to be found for such entities both within memory and in files. It was decided that only one character would be in memory at a time as the amount of memory required to store a whole font would be great. This meant that the file structure had to be able to be updated quickly. The polygons consisted of a number of points in the Cartesian plane i.e. two numbers could represent each point. Using short integers provided adequate accuracy.

The file representation of a font is as follows. The file begins with two hundred and twenty four four byte file offsets - one pointing to the beginning of the definition of each character. A zero pointer indicates an undefined character. The first pointer is for character 040, the second for character 041 etc.. The offsets are in characters from the beginning of the file. Each character definition consists of a sequence of two two byte (or short) integers representing points, preceded by a four byte value giving the length of the definition in bytes. Each point is connected to the previous point except that three points have special meanings. One means the current polygon is ended and the last point should be connected to the first point in the polygon. The second means the polygon is ended but it is an open polygon so no further connexion should occur. The final special point indicates the end of the character definition. The previewer ignores open polygons as these cannot be filled. They are allowed because they are often created when building up a character in the editor.
Character Representations

In memory a character is represented as a singly linked non-circular list of doubly linked possibly circular lists. Each node of the singly linked list represents a polygon and contains a pointer to it's successor and to a doubly linked list. Each node of the doubly linked lists represents a point and contains a pointer to it's predecessor, it's successor and to it's parent node in the singly linked list as well as two integers giving the point's position. If a doubly linked list is circular it represents a closed polygon. If it is non-circular it represents an open polygon and in this case the parent node in the singly linked list points to the start of the list. This structure is maintained by a suite of routines in the struct.c file. Routines in the io.c file handle the transfer of characters to and from files and memory. When a new character is defined it is added to the end of the file. If a old character is changed, it will be replaced in it's old position in the file unless it's definition is now larger. Then it is copied at the end of the file. Compaction replaces all the characters in their original order and removes any old copies.

3.2.2. Programme Organisation

The mainline of the programme and the definition of all globals is contained in the main.c file. This routine opens appropriate files, calls an initialisation routine and then enters a loop which repeatedly calls the interpreter until it returns false. The initialisation and other utility functions are defined in the file util.c. The interpreter takes as it's argument a line of input from the keyboard and enters an appropriate mode of operation until further input from the keyboard becomes available. The mode is decided
from the first non-blank character found in the line. The routine then enters a loop repeatedly reading the graphics tablet, partially processing the information gained and then calling a some routine depending on the current mode. The partial processing done involves reading the puck buttons and using them to update and array of points. This array represents the last several points "marked" by the user and these points are at the same time shown as small circles on the display window. The blue button marks a new point, while the yellow button will mark a point already existing close to the puck's position. If there is no point near the puck the yellow button gives an error message. The white button unmarks the last marked point. The green button serves to confirm that the marked points are correct. The array of marked points and variables indicating how many points are marked, whether they are confirmed and if this is the first call of a new mode are available to the functions which implement each mode.

The functions perform such operations as join two points with a line, remove a line joining two points, insert a point between two connected points, delete a point, draw an arc, move a point etc. These routines use information from the marked point array and other variables to make changes to the current character structure by calling routines from the suite in the struct.c file. They also preview the changes about to be made on the screen. Once the changes are made in the data structure what is previewed is left on the screen.

The "new", "backup" and "exit" functions perform the transfer between the file version of a font and the memory version of one of the characters in that font. This is done by calling the suite of routines in the io.c file. Backup forgets the changes made to the current character and exit terminates the programme after saving the current font. New is the only function which reads the keyboard itself. It reads a character number in octal optionally followed by a two character font name and then saves the current character and reads the new one. There is no default mode in the editor so when these functions are complete, a new mode must be entered from the keyboard before editing can continue.

In addition to a data structure for the current character, there is also an identical buffer structure. This can be manipulated by more advanced commands which perform such operations as move or copy a polygon to the buffer, empty the buffer, translate, enlarge or rotate the contents of the buffer and move the contents of the buffer back to the current character. This allows shapes to be moved between characters or to be transformed in some way. These functions, in addition to using the struct.c suite, also use a suite of routines in the file buffer.c.
There remain two simple modes. The first is the "help" function which prints a command summary on the calling window. The other is the "scale" function which rescales the tablet. This can be done because the cursor position is under programme control. Thus a part of the processing that is done after the tablet is read is to use several variables to transform the puck position into the cursor position. The scaling routine changes these variables to appropriate values. The functions which implement each mode, along with the interpreter, form a suite of routines stored in the graph.c file.

The constant definitions and external declarations for each file except main.c are stored in the files font.h and windat.h. Windat.h contains definitions which are used also in generating the window file. The file disp.c is not a C programme but specifications for the window file generator. The C preprocessor is used to control the values so that they match those known to the programme.

3.2.3. Data Security

Each of the font files represents a large amount of valuable data. It is important that this data be secure against hardware and software failure. For this reason only one character of a font is stored in memory at any time and the font currently being edited is kept in the file tmp in the font file directory. If the system fails during editing or the programme terminates abnormally for any reason, this tmp file will contain a copy of the last font edited with the changes intact except the changes to the last character which will have been lost. This file can be moved over the old copy of the appropriate font file to restore the data. Normally the tmp file will be removed on exiting the editor.

3.3. Preview

3.3.1. Input Format

The format of the document files used by the previewer programme is the format which comes directly from the /usr/lib/troff/rtroff programme on the Perkin Elmer computers. This is a modified version of troff. The format produced is as follows. Each character (octal numbers 000 to 377) represents a command. Characters from octal 040 and above represent a command to set the corresponding (usually ascii) character in the current font and at the current position and point size. Several characters in the range 000 to 037 octal represent commands to change fonts, change point size, move vertically or horizontally, start or stop ruling a line, to end the current page or to initialise. Several of these are followed by several bytes of data, e.g. the movement commands are followed by four bytes which represent a signed integer giving the amount of motion.
required. The font change is followed by two characters of font size and the size change by a single character giving the new point size in the range 5 to 72. Notice that page length and width are completely arbitrary and indeed may be varied throughout a document. For the purposes of page numbering the previewer keeps an idea of the page length (which may be altered interactively but which defaults to eleven inches) and will stop searching for new characters for a page when it finds a character whose relative position places it beyond the page end as determined by this length. The end of page command is ignored by the previewer.

The previewer reads not only the document file but also uses the font definition files. The format of these is given in description of the edfont programme design. If a character is undefined in the current font, the previewer searches for it in the TR or Times Regular font. This device permits the saving of much storage space as there are many characters (such as the Greek character set) which are identical in every font. If it is not defined in the TR font then it is treated as blank.

3.3.2. Internal Data Structures

The main data structure used is the one which maintains the pixel maps of the characters. The conversion from the font file definition to a pixel map is a C.P.U. expensive operation and so characters, once converted, are saved for future use. Due to the large amount of storage required by the pixel maps, a limit had to be imposed on the number of saved characters. This limit was set at 224 characters or a full font. The data structure employed was a simple array containing the font name and point size of the character to which it pointed. Each time a character is required of a different size or font from the current character of the same value, the old definition is be overwritten. The pixel maps themselves are simply a character vector obtained dynamically from the operating system using the rectangle (see RECTANGLE(3)) function. Information required for their use in the system call to draw them on the window, such as the scan line increment (see WRASOP(2)) are also stored in the main array.

A secondary structure used was employed to quickly locate the beginnings of pages. This is simply an array containing the file offsets in the document file of the beginning of the definition of as many pages are known. When the end of a page being previewed is reached, the position of the beginning of the next page is recorded in the array. When the user requests that a page whose start position is not yet known be previewed, the programme reads through the document file recording start positions as it goes, until it reaches the requested page. Because this data depends on the current idea of page length, if the page
length is changed by the user the array is emptied and the
new positions must be recalculated.

3.3.3. Programme Structure

As with edfont, the main.c file contains some simple
initialisation such as reading the command line argument,
opening files and setting variables to their default values.
It then calls the interp routine repeatedly until it returns
false. Interp is the command interpreter. Some commands,
such as exiting the programme, changing the current scale or
page length or width do not redraw the screen but change the
value of status variables. The remaining commands involve
repositioning the preview window within the document and so
all cause the screen to be redrawn.

The drawing routine is a loop which reads commands from
the document file and executes the commands by drawing sym­
bols to the window. It terminates when it finds a character
beyond the end of the page. Interp and drawpage along with
their support routines form a suite in the file interp.c.
These call routines from two other suites — print.c and
raster.c.

The print.c suite contains functions which convert the
image on the window into a file containing data in a format
compatible with Apple's Imagewriter printer. A background
process is then begun using the system (see SYSTEM(2)) sys­
tem call to send this file to the rs232 serial serial port
on the Perq. The shell script for this process is in a file
called screendump. Because of difficulties in handshaking
with the printer's buffer and resulting buffer overflow
problems, the printer must be set to run at 2400 baud. When
the file has been printed, the background process deletes
the file (which is called printfile and is created in the
current directory of the caller). A new printfile cannot be
created until the last one has been deleted so only only
page may be queued for the printer at a time although one
the printing begins, the user is free to continue previewing
the document. The command to initialise printing is p.
The *raster.c* suite contains the *raster* function which converts a polygon character definition into a pixel map. Where characters are to be printed larger than 18-point, this is done using the following algorithm. The pixel map is scanned horizontally one line at a time. Each time a line from a polygon is crossed, the colour of the pixels is changed. Where smaller characters are involved, they are first converted using the same algorithm as though they were eight times larger i.e. with sixty four pixels for every one they need. The number of pixels set to black in each block of sixty four is then counted and if it is greater than a certain constant, the corresponding pixel will be set on a smaller pixel map. In this way even small characters become legible if somewhat rough in appearance. The reduction algorithm could be improved but at the cost of more C.P.U. time.

The file *disp.c* is input to a window generation programme as with the *edfont* programme. It again uses the C preprocessor to read constants from the *windat.h* file. These constants are also available to the code along with the remainder of the constants and external variable definitions through the *preview.h* file.

4. Installation

The makefiles for both *edfont* and *preview* require two constants to be set before *make* is called to install the programmes. These are the *hmedir* constant which should be set to the name of the directory in which the object file and display window files are to reside and the *fntdir* constant which should be set to the name of the directory in which the font definition files reside. *Fntdir* should match
in the makefiles of edfont and preview if they are to access
the same font files. version with enlarging and reducing
facilities. More fonts may be digitised if time allows.

5. Bibliography

The references to programming manuals refer to I.C.L.'s
"Guide to P.N.X." and Bell Labs' "Unix Programmer's manual"
both of which are supplied as documentation for the Perq
computer. The data format for the Apple ImageWriter is
explained in the "Imagewriter User's Manual" as is the
method for setting the baud rate of that machine. The
"Nroff/Troff User's Manual" is a part of section 2A of the
"Unix Programmer's Manual".
Phototypesetter Previewer Technical Report

by James Ashton

University of Wollongong

ABSTRACT

This document gives the technical specifications of a phototypesetter previewing system for the I.C.L. Perq microcomputer. It fully explains not only what functions the system will perform and how these facilities can be used, but also what basic methods are employed in their implementation. Some details concerning installation are also given.

November 4, 1984
Phototypesetter Previewer Technical Report

by James Ashton

University of Wollongong

1. Introduction

Because the phototypesetting programme is not implemented on the Perq, the documents must be prepared on the Computer Science Department's Perkin Elmer computers and transferred to the Perq via a serial line. This requires a number of small support programmes (see 2.1).

The encrypted form of the fonts used by the phototypesetting device means that they are unavailable in a machine readable form and must therefore be manually digitised. The first of the two major programmes performs this task using the Perq's graphics tablet (see 2.2).

The second major programme combines the document file with the font definition information to produce the previews in an interactive manner (see 2.3).

2. Overview

2.1. Document Production and Transfer

Documents for phototypesetting are prepared using a text editor in a format compatible with nroff and troff (see the "Nroff/Troff User's Manual"). To preview such documents, the following command should be executed.

```
troff mcs -R -t document ^ bin > file
```

The -R option to troff specifies that output will be in a relatively compact but device independent form*. Unfortunately raw output is of a binary nature (i.e. eight bits per character are used) and cannot easily be directly transferred along the existing serial line. The bin filter converts this data to an ascii format in a similar way od (see OD(1)) does, but more compactly and with the addition of checksum information. The cu (see CU(1)) command can

* Normally, output (when using the mcs option) is piped through two filters but the -R (or raw) option prevents this as output from the first filter is too verbose for transfer via a slow serial line and output from the second is tailored for a particular device.
then be used to transfer the resulting file to the Perq where it must be again filtered using unbin to restore it to a binary format. This is done with the command

```
unbin < file > file2
```

The resulting file may now be previewed.

### 2.2. Font Editing

The phototypesetter uses a number of fonts (currently eighteen) each of which has a two character ASCII name and may contain up to two hundred and twenty four characters (octal 040 to 377). The phototypesetter previewer requires representations of these fonts to be available on the Perq.

---

The Perq's fonts are derived from listings of characters produced on the phototypesetter. These data must first be digitised and then the digitised data transformed to obtain a usable representation. The first step in digitising is to phototypeset all the required characters as large as possible (currently seventy two point size) with a small marker such as a five point full stop at the lower left and one other corner of the enclosing box around each character. Thus if seventy two point characters are being used, the two points should be one inch apart independent of the width of each individual character. The points are for reference purposes (see 3.2.2).

There are seventy two points per inch so a seventy two point character fits within a one inch square. It may be helpful to further enlarge the characters.
The character images can now be affixed to the graphics tablet (taking care to ensure they are square with the tablet edges) for digitising. Because the tablet and multiple windows are in use, winit (see WINIT(1)) must be called before editing can begin.

The digitising programme is edfont and should be called without argument. When the editing window appears on the screen and three small circles appear on it, the window should be selected using the puck’s yellow button in the normal way. Messages will appear on the window from which the programme was called. Edfont will begin by editing character 040 octal in the TR or Times Regular font. The character and font may be changed interactively at any time. Each font is stored in a file with the same name as the font’s name in a fixed directory. This directory may be changed during installation by changing a constant in the makefile file. New font files are created as required.

Font editing proceeds using both the four coloured keys on the graphics tablet’s puck and the keyboard. The keyboard is used to enter single character commands, one per line, which control the mode of the puck’s operation. In the mode to change characters or fonts, the new character and font are entered from the keyboard. In the different modes, each of the puck buttons retains a very similar function.

The characters are defined as a number of polygons, thus the editor is concerned with generating and connecting points rather than in controlling each pixel in a grid. The characters seen on the screen are therefore shown in outline only. To digitise a character, it is first necessary to change to the appropriate font and character number. Then the puck must be scaled so that the position and size of the character fixed to the tablet map correctly on to the screen. This is done with the aid of the reference points discussed previously. Each time a new character is selected, the old one is saved. Each time a new font is selected, the font is compacted and replaced in a file with an appropriate name — fonts are held in a temporary file during editing. The new font is then copied to the temporary file. The compacting may take considerable time if the font has many characters defined in it.

2.3. Previewing

Like the font editor, the previewer can only be run after winit has been called. The previewer is called using an enlarging photocopier or similar device as the larger the characters are, the easier they will be to digitise.
preview and requires the name of the document file as it's only argument. Once called it will create a window covering almost all of the screen. This window should not be selected as commands are read from the window from which preview was called.

Commands are entered in a style similar to ed (see ED(1)). Decimal numbers with optional fractional parts are entered one per line. By default these request a preview of the page number given (numbered from zero) although by prefixing the number with various letters it is possible to change the page length or width, the scale at which previewing occurs and the horizontal position of the previewing. The letter "p" by itself will copy the current screen to a printer attached to the serial line. If a message indicating that "printfile" cannot be created appears, this indicates that too many previous printer jobs are queued. The command "x" terminates the programme.

The previewer knows the name of the directory in which the font definition files can be found. If this directory is changed during the installation of edfont, preview should be installed with a similar change made to it's makefile. If a font or character cannot be found, a definition from the TR font is substituted. The font files contain definitions in the form of polygons. There are a number of routines which convert these polygons to grids of pixels using scaling as required by the point size indicated in the document file. In this way a single font file can be used to represent the same font at a large range of sizes. While the output quality suffers, this is considered acceptable considering the storage savings made and the previewing nature of the output.

3. Design Details

3.1. Bin and Unbin

These two programmes are simple filters which read from standard input and write to standard output. They function identically except that unbin performs the inverse of bin and at the same time performs checksum verification. Bin takes a binary file and converts it to a file containing characters in the set "0" - "9", "A" - "F", "\n" and "X". Each line contains sixty four hexadecimal digits representing thirty two digits from the original file, followed by four hexadecimal digits which are the checksum of the characters. The last line may contain "X"s to fill the space between the last character's representation and the final checksum. Both bin and unbin use buffering to speed their

* The white key on the puck can be used to alternately view the display and calling windows as required.
The polygon data structure is a list of lists. Each node of the lower level list contains the coordinates of the point it represents.

Character Representation is copied at the end of the file. Compaction replaces all the characters in their original order and removes any old copies.

3.2.2. Programme Organisation

The mainline of the programme and the definition of all globals is contained in the main.c file. This routine opens appropriate files, calls an initialisation routine and then enters a loop which repeatedly calls the interpreter until it returns false. The initialisation and other utility functions are defined in the file util.c. The interpreter takes as it's argument a line of input from the keyboard and enters an appropriate mode of operation until further input from the keyboard becomes available. The mode is decided from the first non-blank character found in the line.

The routine then enters a loop repeatedly reading the graphics tablet, partially processing the information gained and then calling a some routine depending on the current mode. The partial processing done involves reading the puck buttons and using them to update an array of points. This array represents the last several points "marked" by the user and these points are at the same time shown as small circles on the display window. The blue button marks a new point, while the yellow button will mark a point already existing close to the puck's position. If there is no point near the puck the yellow button gives and error message. The white button unmarks the last marked point. The green
button serves to confirm that the marked points are correct. The array of marked points and variables indicating how many points are marked, whether they are confirmed and if this is the first call of a new mode, are available to the functions which implement each mode.

The functions perform such operations as joining two points with a line, removing a line joining two points, inserting a point between two connected points, deleting a point, drawing an arc, moving a point etc. These routines use information from the marked point array and other variables to make changes to the current character structure by calling routines from the suite in the struct.c file. They also preview the changes about to be made on the screen. Once the changes are made in the data structure what is previewed is left on the screen.

The "new", "backup" and "exit" functions perform the transfer between the file version of a font and the memory version of one of the characters in that font. This is done by calling the suite of routines in the io.c file. Backup forgets the changes made to the current character (after prompting for and receiving a "y" from the keyboard for confirmation). Exit terminates the programme after saving the current font. New and backup are the only functions which read the keyboard itself. New reads a character number in octal optionally followed by a two character font name and then saves the current character and reads the new one. There is no default mode in the editor so when these functions are complete, a new mode must be entered from the keyboard before editing can continue.

In addition to a data structure for the current character, there is also an identical buffer data structure. This can be manipulated by more advanced commands which perform such operations as moving or copying a polygon to the buffer, emptying the buffer, translating, enlarging or rotating the contents of the buffer and moving the contents of the buffer back to the current character. This allows shapes to be moved between characters or to be transformed in some way. These functions, in addition to using the struct.c suite, also use a suite of routines in the file buffer.c.

There remain two simple modes. The first is the "help" function which prints a command summary on the calling window. The other is the "scale" function which rescales the tablet. This can be done because the cursor position is under programme control. Thus a part of the processing that is done after the tablet is read is to use several variables to transform the puck position into a cursor position. The scaling routine changes these variables to appropriate values. The functions which implement each mode, along with the interpreter, form a suite of routines stored in the
graph.c file.

The constant definitions and external declarations for each file except main.c are stored in the files font.h and windat.h. Windat.h contains definitions which are used also in generating the window file. The file disp.c is not a C programme but a specification file for the window file generator. The C preprocessor is used to control the values so that they match those known to the programme.

3.2.3. Data Security

Each of the font files represents a large amount of valuable data. It is important that this data be secure against hardware and software failure. For this reason only one character of a font is stored in memory at any time and the font currently being edited is kept in the file tmp in the font file directory. If the system fails during editing or the programme terminates abnormally for any reason, this tmp file will contain a copy of the last font edited with the changes intact except the changes to the last character which will have been lost. This file can be moved over the old copy of the appropriate font file to restore the data. Normally the tmp file will be removed on exiting from the editor.

3.3. Preview

3.3.1. Input Format

The format of the document files used by the previewer programme is the format which comes directly from the /usr/lib/troff/rtroff programme on the Perkin Elmer computers *. The format produced is as follows. Each character (numbered 000 to 377 in octal) represents a command. Characters from octal 040 and larger represent a command to set the corresponding (usually ascii) character in the current font and at the current position and point size.

Several characters in the range 000 to 037 octal represent commands to change fonts, change point size, move vertically or horizontally, start or stop ruling a line, to end the current page or to initialise. Some of these are followed by data bytes. The movement commands are followed by four bytes which represent a signed integer giving the amount of motion required. The font change is followed by two characters giving the font name and the size change by a single character giving the new size in the range 5 to 72 points.

*This is a version of troff modified by Ross Nealon so that the output format is less device independent.
Page length and width are completely arbitrary and may be varied throughout a document. For the purposes of page numbering, the previewer keeps an idea of the page length (which may be altered interactively but which defaults to eleven inches*) and will stop searching for new characters for a page when it finds a character whose relative position places it beyond the page end as determined by this length. The end of page command is ignored by the previewer, mainly because the version of troff in use does not guarantee to output the command at the end of every page.

The previewer reads not only the document file but also uses the font definition files. The format of these is given in description of the edfont programme design (see 3.2.1). If a character is undefined in the current font, the previewer searches for it in the TR or Times Regular font. This mechanism permits the saving of much storage space as there are many characters (such as the Greek character set) which are identical in every font. If it is not defined in the TR font it is treated as blank.

3.3.2. Internal Data Structures

The main data structure used is the one which maintains the pixel maps of the characters. The conversion from the font file definition to a pixel map is a C.P.U. expensive operation and so characters, once converted, are saved for future use. Due to the large amount of storage required by the pixel maps, a limit had to be imposed on the number of saved characters. This limit was set at 224 characters or a full font. The data structure employed was a simple array containing the font name, point size and other relevant characteristics of the character definition to which it pointed. Each time a character is required of a different size or font from the current character of the same value, the old definition is overwritten. The pixel maps themselves are simply character vectors obtained dynamically from the operating system using the rectangle (see RECTANGLE(3)) function. Information required for their use in the system call to draw them on the window, such as the scan line increment (see WRASOP(2)) are also stored in the main array.

A secondary structure was employed to quickly locate the beginnings of pages. This is simply an array containing the file offsets in the document file of the

* However long the page, only eleven inches can be seen on the Perq screen at a time although with different scaling factors this may represent more or less than eleven inches of document. The length value refers to document length, not display length, and is thus independent of the scale factor.
beginning of the definition of as many pages are known. When the end of a page being previewed is reached, the position of the beginning of the next page is recorded in the array. When the user requests that a page whose start position is not yet known be previewed, the programme reads through the document file recording start positions as it goes, until it reaches the requested page. Because this data depends on the current idea of page length, if the page length is changed by the user the array is emptied and the new positions must be recalculated.

3.3.3. Programme Structure

As with edfont, the main.c file contains some simple initialisation such as reading the command line argument, opening files and setting variables to their default values. It then calls the interpret routine repeatedly until it returns false. Interpret is the command interpreter. Some commands, such as exiting the programme, changing the current scale or page length or width do not redraw the screen but change the value of status variables. The remaining commands involve repositioning the preview window within the document and so all cause the screen to be redrawn.

The drawing routine is a loop which reads commands from the document file and executes the commands by drawing symbols to the window. It terminates when it finds a character beyond the end of the page. Interpret and drawpage along with their support routines form a suite in the file interp.c. These call routines from two other suites called print.c and raster.c.

The print.c suite contains functions which convert the image on the window into a file containing data in a format compatible with Apple's Imagewriter printer. A background process is then begun using the system (see SYSTEM(2)) system call to send this file to the rs232 serial port on the Perq (file /dev/rs). The shell script for this process is in a file called screendump. Because of difficulties in handshaking with the printer's buffer and resulting buffer overflow problems, the printer must be set to run at 2400 baud.

Screendump is a crude spooling programme which must continue to exist as a background process while the printer file it controls awaits printing. It's first action is to give the output file from the preview programme (always called /tmp/printfile) a temporary file name unless three such temporary files already exist. It will then repeatedly try to access the device file to send the file at thirty second intervals until it is successful before removing the file. If three temporary files exist, /tmp/printfile is not renamed so preview cannot print a new screen if too many are
already queued.

Generation of pixel maps from polygons.

Reduction of a character section by counting the number of black pixels in 8 by 8 areas. Only those with 24 or more have their corresponding large pixel set black.

Polygon to Pixel Map Conversion

The raster.c suite contains the raster function which converts a polygon character definition into a pixel map. Where characters are to be printed larger than 18 point, this is done using the following algorithm. First all the polygons constituting the character are broken up into a number of line sequences such that consecutive vertex points in the sequence have increasing y values. An ordered list of the sequences in then made with the sequences with the smallest initial y value are first in the list. It is now simple to determine which lines cross any given horizontal line. The pixel map is scanned one horizontal line at a time. Each time a line from a polygon is crossed, the colour of following pixels in the horizontal line is alternated. Where smaller characters are involved, they are
first converted using the same algorithm as though they were eight times larger i.e. with sixty four pixels for every one they need. The number of pixels set to black in each block of sixty four is then counted and if it is greater than a certain constant, the corresponding pixel will be set on a smaller pixel map. In this way even small characters become legible, if somewhat rough in appearance. The reduction algorithm could be improved but only at the cost of more C.P.U. time.

The file disp.c is input to a window generation programme as with the edfont programme (see 3.2.2). It again uses the C preprocessor to read constants from the windat.h file. These constants are also available to the code along with the remainder of the constants and external variable definitions through the preview.h file.

4. Installation

The makefiles for both edfont and preview require two constants to be set before make is called to install the programmes. These are the hmedir constant which should be set to the name of the directory in which the object file and display window file are to reside, and the fntdir constant which should be set to the name of the directory in which the font definition files reside. Fntdir should match in the makefiles of edfont and preview if they are to access the same font files.

5. Bibliography

The references to programming manuals refer to I.C.L.'s "Guide to P.N.X." and Bell Labs' "Unix Programmer's manual" both of which are supplied as documentation for the Perq computer. The data format for the Apple Imagewriter is explained in the "Imagewriter User's Manual" as is the method for setting the baud rate of that machine. The "Nroff/Troff User's Manual" is a part of section 2A of the "Unix Programmer's Manual". As always, when programming in C, Kernighan and Ritchie's "The C Programming Language" is invaluable.
Edfont User's Manual

by James Ashton

University of Wollongong

ABSTRACT

This document describes the basics of using edfont on the I.C.L. Perq computer. Edfont is a utility which allows the manipulation of font definition files used by the phototypesetter pre-viewer. Operation of all the different command modes is described in enough detail to allow an inexperienced user to become proficient in it's use.

November 4, 1984
Edfont User's Manual

by James Ashton

University of Wollongong

1. Introduction

Edfont is an interactive font editor for creating and modifying font definition files. It takes input from two input devices, the console keyboard and the graphics tablet. Use of these is described in the "Guide to P.N.X.".

1.1. Getting Started

The first thing to do is to enter the window manager utility winit (see WINIT(1)). This provides a convenient environment for edfont, allowing many display windows to be active at once. Edfont uses two windows. The first is a simple text window which must be created by the user using winit. Select this window so that a "$" prompt appears in it and give the command "edfont". Edfont will now automatically create a graphics window in which the editing will be done. This should now be selected until a cross cursor appears. The default character will be read from the font files. This is the Times Roman 040 (octal) character. Before any editing can begin, a command mode must be entered.

2. The "New" command

Commands are entered from the keyboard by hitting a single key followed by the return key. The key hit is usually the first letter of the command's name. The "new" command selects a new character and possibly a new font. Enter "n" and on the next line an octal number in the range 040 to 377. The character being edited will now be saved and the newly named character will be displayed on the screen. An option of the new command is to change fonts at the same time. To do this follow the octal number by a space and the two character font name on the same line. Because font files can be large and changing fonts requires the saving of the old one, there can be a considerable time taken by edfont when a new font is selected (or the programme is exited). An important point to note is that to avoid ruining the graphics display, the characters entered from the keyboard are not displayed on the screen. Messages are sent to the text window so it may be useful to place this window
at the bottom of the screen, below the position of the graphics window, so that the text can be seen.

3. The "Scale" command

When the desired character is reached using \texttt{new}, the tablet must be scaled so that it may be digitised in the appropriate position relative to the baseline, and at the correct scale. The copy of the character you intend to digitise should be as large as possible and should have marked on it the position of the left end of the baseline and another point a fixed distance away. This distance represents the point size of the character so if a seventy two point font is being digitised, the point should be horizontally or vertically separated from the left end of the baseline by seventy two points (or one inch).

Scale needs to know these two points' position so move the puck until the cross is over the first (left end of the baseline) point and press the blue key. A small circle should appear on the screen. Mark the second point in the same way. You may delete the marked points using the white key. Once you have them correctly positioned, press the green key to confirm that they are correct. The cursor will now be correctly scaled to the puck for digitising the character.

4. The "Line" command

The line command is entered using the letter \texttt{l}. Mark a point with the blue key and you will find that now a line joining the marked point and the cursor will be created. Mark a second point and the line will join the two marked points. The green key can be used to make the line permanent. It will also have the effect of creating a second line joining the second marked point and the cursor position. By using the blue and green keys alternately a series of connected lines may be drawn. To begin a new sequence of lines, use the white key to delete any remaining marked points.

The coloured keys on the puck operate in a consistent fashion throughout each mode. The blue key marks new points while the white key deletes them. The green key confirms that the marked points are in the correct position so the changes made can become permanent. The yellow key is used for marking points which have already been made permanent. For example to extend a series of lines by adding a new line to the end, first move the puck until the cursor is near the end point of the end line and press the yellow key. The point is now remarked and can be used in the same way as any other marked point.

It is important to realise that the editor is dealing
with polygons and not with graph structures. For this reason only two lines can end at any one point. An attempt to attach a line to a point with two lines already attached will result in an error message.

5. The "Unline" command

This command is entered using the letter "u" and reverses the effect of the line command. Use the yellow key to mark the endpoints of an existing line. The points marked must be connected directly or an error message will result. This line will now be deleted.

6. The "Insert" command

This command inserts a new point between two existing connected points. Mark the existing points first (with the yellow key) and the connecting line will be broken and two lines from the existing points will join at the cursor. Now mark a third point and use the green key to confirm the change.

Edfont has a buffer of four points so up to four points may be marked at a time for use in the commands. If more than four are marked, the old point four will be move to the new position.

7. The "Delete" command

The delete command has the command letter "d". Once the point to be deleted is marked with the yellow key and confirmed, it will be removed. Any connexions to it will be broken.

8. The "Fix" command

This command moves an existing point to a new location without breaking the connexions to it. The new location may not be an already existing point. It is entered using the letter "f".

9. The "Arc" command

Like the line command the arc command joins two marked points but uses a series of lines to do so to create an elliptical arc. Care must be taken where points along the arc are near existing points as it may become difficult to distinguish between them. The first point marked will always have a horizontal tangent.

10. Buffer Usage
10.1. Putting data in the buffer

Edfont uses a buffer to allow more advanced operations to be performed on whole polygons at once. The first step is to move or copy (using the "Move" and "Copy" commands entered with the letters "m" and "c") to either move or copy a whole polygon from the current character to the polygon buffer. The buffer is not displayed on the screen so if a move is performed rather than a copy, the polygon will disappear from the screen. Polygons are marked by marking any one of their vertex points.

10.2. Modifying the buffer

Three functions modify the buffer. The simplest is the `translate` command entered using the letter "t". Mark one point and the polygons in the buffer will appear on the screen. By moving the puck, the whole buffer will be moved until a second point is marked. Confirming the two points with the green key will make the translation permanent and the buffer will again disappear from the screen. The other two commands, "enlarge" and "rotate", operate in a very similar manner.

10.3. Deleting the buffer

The command `put` moves the buffer back to the current character leaving the buffer empty. The command `zap` deletes the buffer. These are the only two commands which remove polygons from the buffer. The buffer can contain any number of polygons and be used to transfer polygons between characters by copying the polygons, doing a `new` command followed by a `put` command.

11. Deleting changes

Two commands are useful if unwanted changes have been made. The `backup` command goes back to the version of the current character that existed before the last new command was executed. To be safe it first prompts for a "y" from the keyboard before proceeding. The `kill` command is similar but it entirely removes any definition of the current character that may exist.

12. Exiting

To exit from edfont, use the `exit` command. This is the only command where the command letter ("x") is not the first letter of the command. If the current font contains many characters, the programme may take up to a minute to exit. This process should not be interrupted as corruption of the font file directory could result.
13. The font directory

The font directory is `/usr/lib/fonts`. It contains files with two character font names. When edfont is working with a font, it makes a copy of the appropriate file in `/usr/lib/fonts/tmp`. When it changes fonts or exits, it compacts the temporary file back to its original location. If for any reason edfont terminates abnormally, it may be possible to retain the changes made to date by moving the temporary file to the correct location (i.e. to rename it with the required font name). It is suggested that this not be done until the contents of `/usr/lib/fonts/tmp` have been checked by renaming it with an unused font name and editing that font.

14. Command summary

The capital letter in the command name indicates the letter by which it is entered.

Arc  Connects two points with an elliptical arc.
Backup Backs up to the last version of the current character to be saved.
Copy Copies a polygon to the polygon buffer.
Delete Deletes a point.
Enlarge Enlarges the polygon buffer.
Fix Moves a point.
Help Prints a help screen summary.
Insert Inserts a new point between two existing points.
Kill Removes any definition of the current character in the current font.
Line Joins two points with a straight line.
Move Move a polygon to the polygon buffer.
New Changes the current character and optionally the current font.
Rotate Rotates the polygon buffer.
Scale Rescales the tablet.
Translate Translates the polygon buffer.
Unline Deletes a connecting line between two points.

Verify Prints a listing of the coordinates of the points in the current character.

Exit Exits from edfont.

Zap Deletes the polygon buffer.

15. Puck key summary

Blue Marks a new point.

White Unmarks the last point marked.

Yellow Marks an already existing point.

Green Confirms the marked points as correct.

16. Warnings

Use of the buffer can be dangerous if a point in the buffer is in the same position as an already existing point. These points are then difficult to distinguish. Because all lines are drawn by toggling pixels on the screen, lines which coincide will disappear. This can be confusing as the lines remain in existence in the current character but cannot be seen. The only way to detect this is using the "verify" command which lists all the points currently defined.

17. Conclusions

Edfont provides a reasonably high level environment for editing polygonal character definitions. With experience digitising can become quite rapid. The only way to become familiar with the command modes is through experimentation with them.

18. Bibliography

A useful guide to general usage of the Perq's keyboard and graphics tablet and facilities in general is the "Guide to P.N.X.". As the system is an implementation of system seven Unix, the "Unix Programmer's Manual" can also be helpful. Both of these volumes are supplied as documentation for the I.C.L. computer on which edfont runs.

*
Preview User's Manual

by James Ashton

University of Wollongong

ABSTRACT

This document describes the basics of using preview on the I.C.L. Perq computer. Preview is an interactive programme which allows the user to see simulated phototypesetter output. This manual should serve as an introduction to its use for inexperienced users.

November 4, 1984
1. Introduction

Preview allows the simulation of the phototypesetter given an input file of the correct format. Such files may be obtained from the output of troff using the "-R" option (see TROFF(1)). Because troff is unimplemented on the I.C.L. Perq computer, on which preview runs, it will be necessary to transfer the troff output to the Perq using a serial line (see CU(1) and BIN(1)).

2. Getting Started

The document to be previewed should be processed using the following command.

```
troff mac -R -t file > bin > file2
```

File2 should then be transferred to the Perq using cu and the following command run.

```
unbin < file2 > file
```

Before preview can be run, winit must be entered (see WINIT(1)) and a window created and selected. Finally preview can be called with:

```
preview file
```

The window that preview automatically creates covers almost the whole screen so that the window from which it was called cannot be seen. To see the commands that have been entered as well as the error and status information sometimes output by preview, this window can be brought to the foreground, as described in the "Guide to P.N.X.", with the white key on the puck. Preview operates using single line commands in a simple format.

3. Page numbering

Pages are numbered from zero. The simplest command that can be given is just to press the return key. This results in printing the next page. If this is the first command given, page zero is printed. To jump to any page in the document, type the number of that page (in decimal) and press return. It is also possible to move to a new page relative to the current page. This is done by prefixing the number by a '+' symbol to move forward or a '-' symbol to move backwards. Fractional page numbers are also possible.
To print the bottom half of page ten, for example, type "10.5". The previewing always stops at the bottom of the page on which it begins so in the example given, none of page eleven will be shown.

4. **Horizontal movement**

Sometimes it may be necessary to view documents which are wider than the Perq's screen. To do this prefix a "h" to a number in the same format as discussed above. The current page will then be displayed as many pages offset from the left margin as specified by the number. Again, relative and fractional numbers are allowable.

5. **Page size**

The previewer keeps an idea of the page length and width which begins as eleven inches and eight inches respectively. These figures may be changed using the letters "l" and "w" prefixed to numbers as before. The units understood are inches. When the length is changed, the paged numbering scheme is, of course, changed.

6. **Scaling**

To allow the viewing of fine details or of very large pages, preview can operate at five different scales from 0 to 4. The default scale is 2 which means a one to one scale. Scale 4 enlarges two-fold and scale 0 reduces two-fold. By prefixing the letter "s" to a number in the range 0 to 4, the current scale can be changed.

7. **Printing**

A copy of the current screen image can be sent to an Imagewriter printer. Up to four screen images can be queued at once but if more than one page is queued, the order they are printed in is not guaranteed to be the same as the order in which they are submitted. The command to do this is the "p" command. The printer should be set to 2400 baud and connected to the Perq's serial port.

8. **Miscellaneous information**

The "?" command prints out the current screen position, page size and scale. The "v" command means the same as just a number alone and means change vertical position. To exit from preview, use the "x" command.

9. **Command Summary**

Print status information. Move horizontally. Change to a new page length in inches. Print the current screen on an Imagewriter printer. Move vertically. If no number is given it is assumed to be +1. Change to a new page width in inches. Exit from preview.
10. Conclusions

Preview is a simple utility to use, with most of the difficulty lying in creating and transferring the document files required by it. Due to the complexity of its operation it is unfortunately slow. The quality of its output is not intended to be of high quality. An attempt to make it so would result in a further drastic speed loss. Output should be adequate for most error correcting purposes. Characters down to eight point are legible at scale 2 and at this scale too a full A4 page can be previewed on the screen.

11. Bibliography

A useful guide to general usage of the Perq facilities in general is the "Guide to P.N.X.". As the system is an implementation of system seven Unix, the "Unix Programmer's Manual" can also be helpful. Both of these volumes are supplied as documentation for the I.C.L. computer on which Preview runs. Information about the Imagewriter printer and setting its baud rate can be found in the "Imagewriter User's Manual".
NAME
bin, unbin - font editor

SYNOPSIS
bin, unbin

DESCRIPTION
These commands convert files to and from eight bit binary and ascii formats. Both read from standard input and write to standard output. They are useful for copying files via seven bit data paths such as some serial lines.

Bin takes binary data as input and outputs ascii characters in the set '0' to '9', 'A' to 'F', '\n' and 'X'. Output is a hexadecimal dump of the input with thirty two bytes becoming sixty four hexadecimal digits followed by four checksum digits on a line. The 'X' character acts as a place marker on the last line.

Unbin performs the reverse conversion making checksum checks. It does not terminate if errors are detected but gives a warning on the error output.

AUTHOR
James Ashton
University of Wollongong

SEE ALSO
cu(1)

DIAGNOSTICS
Unbin produces a single line message for every checksum error detected.

BUGS
The conversion is inefficient, taking two ascii characters per eight bit byte. There should be an option to turn off checksum production and checking.
NAME
edfont - font editor

SYNOPSIS
edfont

DESCRIPTION
Edfont is an interactive font editor for the I.C.L. Perq graphics computer. It takes input from both the console keyboard and the graphics tablet device and manipulates font definition files.

With few exceptions keyboard input is limited to one character per line commands which set the operating mode. Puck input involves all four of the puck's buttons to perform functions which are similar in all modes of operation.

The characters are defined as a number of polygons so editing involves the manipulation of vertex points and the lines which connect them. Most command modes perform their function using a number of points in a small buffer of four points which is controlled by the puck. These points are known as marked points.

The puck button function are as follows.

blue       Adds a new point at the current position of the puck to the point buffer.
yellow     Adds a point already defined in a polygon to the point buffer. The point nearest the puck's current position is used but it must be within ten pixels of that position.
white      Deletes the point most recently added to the point buffer from the point buffer.
green      Confirms the positions of all the points in the point buffer.

The operating mode is set each time a line is entered and depends only on the first non-whitespace character on the line. Case is ignored. The operation of each mode is listed below with the capital letter in each name indicating the command character.

Arc         Connects the first two marked points.
Backup      Throws away all the changes made to the current character after prompting for a 'y' from the keyboard for confirmation.
Copy
Copies the polygon of which the first marked point is a vertex to the polygon buffer.

Delete
Deletes the first marked point from the character. If this point is connected to two other points, the connexions are broken and the two points joined directly.

Enlarge
Enlarges the contents of the polygon buffer in both the horizontal and vertical directions an amount proportional to the horizontal and vertical distance between the first and second marked points. The enlargement is about the position of the first marked point.

Fix
Changes the position of the first marked point to the second marked point.

Help
Prints a command summary on the window from which edfont was invoked.

Insert
Disconnects the first and second marked points (which must have been connected) and connects the third marked point between them.

Kill
Removes the definition of the current character after prompting for a 'y' from the keyboard for confirmation.

Line
Connects the first and second marked points.

Move
Moves the polygon of which the first marked point is a vertex to the polygon buffer.

New
Saves the current character and reads the octal number of a new character optionally followed by a two character font name. The new character is then read in and displayed.

Put
Moves the contents of the polygon buffer to the current character.

Rotate
Rotates the contents of the polygon buffer at an angle determined by the position angle of the first and second marked points.

Scale
Rescales the tablet so that the position of the first marked point becomes the bottom left of the editing window and the distance between the first and second marked points becomes the length of the edge of the editing window.
Translate Translates the contents of the buffer a distance and direction equal to the distance and direction separating the first two marked points.

Verify Prints a list of the actual coordinates of the points in the current character.

exit Terminates the programme after saving the current character and font.

Zap Deletes the contents of the polygon buffer.

Most of these functions preview the operation they are about to perform. The character is not actually updated until the points are confirmed using the green button. The polygon buffer is not displayed unless some operation is being performed using it.

The font files are transparent to the user. They are compatible with and in a location known to the troff previewer.

AUTHOR
James Ashton
University of Wollongong

FILES
/usr/lib/fonts/??, font definition files.
/usr/bin/eddisp, editing window.

SEE ALSO
preview(1)

DIAGNOSTICS
The diagnostics produced by edfont are intended to be self-explanatory.

BUGS
There is no default mode. Several modes such as new and scale perform their operation once only and a new mode must be entered from the keyboard before editing can continue.
NAME
preview - troff previewer

SYNOPSIS
preview file

DESCRIPTION
Preview is an interactive phototypesetter previewer which accepts files in a format produced by troff with the -R option (see troff(1)). It runs on the I.C.L. Perq graphics computer.

Commands are read from the console keyboard and take the following form. Case is ignored.

? Displays current status information giving the display's horizontal and vertical position, scale and page length and width.

h[+-]n[.m] Display the current page horizontally offset the given number of page widths.

l[+-]n[.m]

p Queue the current window image for the printer. A maximum of four images may be queued at once and the order of printing is not guaranteed.

s[+-]n Change the scale at which future displays will be shown. Values must be in the range 0 to 4. 2 gives one to one scale and each larger value doubles the size of the display.

v[+-]n[.m] Display page n. If m is given, the page is started the given fraction of a page from the top. This is useful for large scales or long pages.

w[+-]n[.m] Changes the page width to the value given in inches. The initial value is eight inches.

x Terminate the programme.

Where numbers are preceded by a sign, they are evaluated as relative to the current value. The default command is "v" so if a number only is given it is treated as a vertical position. If a blank line is given it is treated as "v +1".

AUTHOR
James Ashton
University of Wollongong
FILES
   /usr/lib/fonts/??, font definition files.
   /usr/bin/predisp, editing window.

SEE ALSO
   edfont(1), troff(1)

DIAGNOSTICS
   The diagnostics produced by prefont are intended to be self-explanatory. If it
   complains that it cannot create a printfile then either too many files are spooled
   for the printer or a prior abnormal termination has left old files still in existence.
   To empty the queue remove /tmp/printfile and files of the form /tmp/pf*.

BUGS
   The previewer is sometimes confused about where the file ends and will continue
to display blank pages beyond the end of the document. The printer spooler is very
primitive and does not guarantee that page order will be maintained.