306-144
Issue 1

5620 Dot-Mapped Display
Release 2.0
Reference Manual

TRADEMARKS

The following trademarks are used in this manual:

- *WE* — Registered trademark of AT&T
- *UNIX* — Trademark of AT&T
- *TELETYPE* — Registered trademark of AT&T
- *TEKTRONIX* — Registered trademark of Tektronix, Incorporated
- *DOCUMENTER'S WORKBENCH* — Trademark of AT&T

CONTENTS

INTRODUCTION

INTRODUCTION

This manual contains reference information for all the user commands and library routines used with the TELETYPE* 5620 Dot-Mapped Display terminal (DMD). A particular manual page may contain references to manual pages located in the *UNIX† System User Manual.*

MANUAL ORGANIZATION

This manual is divided into four sections, some containing sub-classes, as follows:

1. Commands and Application Programs:
   1. General Purpose Commands
   1C. Communication Commands

---

* Registered trademark of AT&T

† Trademark of AT&T

1G. Graphics Commands.

2. Subroutines:
   3C. C and Assembler Library Routines
   3H. Resident in host UNIX Operating System
   3L. Library Routines
   3M. Mathematical Library Routines
   3R. Resident in DMD Read Only Memory
   3S. Standard I/O Library Routines
   3X. Miscellaneous Routines.

3. File Formats.

4. Special Files.

Commands and Application Programs

This section describes programs intended to be invoked directly by the user or by command language procedures, as opposed to subroutines, which are intended to be called by the user's programs.

Subroutines

This section describes the available subroutines, which are intended to be called by the user's programs.

File Formats

This section describes the structure of particular kinds of files; for example, the format of the output of the link editor is given in **dmda.out(4)**.

Special Files

This section discusses the characteristics of each system file that actually refers to an input/output device. The names in this section generally refer to device names for the hardware, rather than to the names of the special files themselves.

INTRODUCTION TO MANUAL PAGE

Manual pages describe user commands and library routines in greater detail. Manual pages use a common format described in the following list. However, some parts of a manual page that do not apply to a specific command may be omitted. The name of the command is in the top corners of the page.

NAME NAME gives the name(s) of the command and briefly states the purpose of the command.

SYNOPSIS

SYNOPSIS summarizes the use of the command (program) being described by the manual page. This part uses special typesetting characteristics to denote items, these characteristics are:

Boldface strings are literals and are to be typed as shown.

Italic strings usually show substitutable argument prototypes and program names found on other manual pages; if underlined, the item is typed as shown.

Square brackets ([]) around an argument prototype indicate that the argument is optional. When an argument prototype is shown as **name** or **file**, the argument prototype always refers to a **file** name.

Parentheses (...) show that the previous argument prototype may be repeated.

Note: Use caution before having a filename beginning with -, +, or = sign. The sign may be taken as a type of flag argument, even when it is in the filename position.

DESCRIPTION
DESCRIPTION describes the subject.

EXAMPLE(S)
EXAMPLE(S) gives example(s) of usage, where appropriate.

SEE ALSO
SEE ALSO gives references to other related information.

DIAGNOSTICS
DIAGNOSTICS describe the diagnostic indications that may be produced. Self-explanatory messages will not be described.

WARNINGS
WARNINGS tell of potential problems.

BUGS BUGS give known bugs and occasional deficiencies. The recommended repair may be described.

DOCUMENTATION AVAILABLE

Below is a complete list of documentation for this product. Select codes for ordering are in parentheses.

- *5620 Dot-Mapped Display User Guide* (306-140)
- *5620 Dot-Mapped Display Administrator Guide* (306-141)
- *5620 Dot-Mapped Display Reference Manual* (306-144)
- *5620 Dot-Mapped Display Product Overview* (306-145)
- *5620 Dot-Mapped Display Application Development Guide* (306-142)
- *5620 Dot-Mapped Display Text and Graphics Application Guide* (306-143)
- *Installation Guide and Release Notes for Core Utitilies Package* (306-146)
- *Installation Guide and Release Notes for Application Development Package* (306-147)
- *Installation Guide and Release Notes for Text and Graphics Package* (306-148)

ORDERING DOCUMENTS

For ordering and pricing information contact the (CIC) Commercial Sales Representatives at this number:

Anywhere 1-800-432-6600 (toll free)

When you have the current prices and are ready to order, refer to the following checklist.

- All orders must be prepaid and are subject to state and local taxes. If your organization is tax-exempt, provide a copy of your exemption certificate in lieu of the sales tax (except in Alaska, Arkansas, Delaware, Hawaii, Illinois, Maine, Montana, New Hampshire, Oregon, South Dakota, and Vermont).

- When ordering from the (CIC) Customer Information Center, please include the following information:

1. The six digit select code of each document.
2. The title of each document.
3. The quantity of each document
4. Ship-to address

- Make checks payable to AT&T Technologies.

- Send your order to:

AT&T Technologies, Inc.
CIC Commercial Sales
Post Office Box 19901
Indianapolis, Indiana 46219

- Orders shipped outside the continental United States must include freight charges. Freight is prepaid for orders shipped within the continental United States. Quotes expire within 30 days of the date of reply.

STRUCTURE DEFINITIONS

MEMORY ALLOCATION

DATA

BITMAPS

GARBAGE COLLECTION

OPERATIONS

ARITHMETIC

INTEGER FUNCTIONS

SORTING

COMPARISON

MOVEMENT

INCLUSION

CONVERSIONS

STRINGS

GRAPHICS

POINTS

LINES

RECTANGLES

CIRCLES

ELLIPSES

BITMAPS

FILLING

MOUSE

COMMUNICATIONS

RESOURCES

KEYBOARD

TERMINAL TO HOST

HOST TO TERMINAL

PRINTER

PROCESS CONTROL

TIME

DATA CONVERSIONS

BIT MANIPULATION AND MASKS

TRIGONOMETRY

ROM ROUTINE ADDRESSES

CONTENTS

NAME

32ld – bootstrap loader for DMD

SYNOPSIS

32ld [**–d**] [**–r**] [**–p**] [**–z**] file

DESCRIPTION

The *32ld* utility loads the named *file* for execution in the DMD connected to its standard output. *File* must be a DMD object file. *32ld* does all necessary bootstrap and protocol procedures. There are several options.

–d causes a printout of the sizes of the text, data, and bss portions of *file* on the diagnostic output.

–p prints down-loading protocol statistics on the diagnostic output.

–r causes *file* to be relocated to an address established by a separate protocol during bootstrap. It is invoked automatically when loading a process into *layers*(1), and usually need not be set explicitly.

–z loads the process but does not run it. It must be started using *dmdebug*(1). It works only under *layers*(1).

The environment variable **JPATH** is the analog of the shell's **PATH** variable to define a set of directories in which to search for *file*.

NOTE: Standard error should be redirected when using the **–d** or **–p** options.

EXAMPLE

Loading bounce into *layers*:

```
32ld bounce
```

SEE ALSO

dmdebug(1), jx(1), layers(1).

NAME

bcan – filter to process DMD 5620 screendumps

SYNOPSIS

bcan [**–i**] [**–x**] [**–r**] file

DESCRIPTION

Bcan is a filter that can be used to format a file originally created using the *screendump* command. The following options are provided:

–i format *file* for the **C.itoh** dot-matrix printer. This output can be sent to the printer to obtain a hard-copy of the bitmap acquired by *screendump*. In order to get the file printed through the lp spooler package, the output from *bcan* must be directed to a file with the suffix "**.g**", and then printed. See *lpg*(1).

–x produce an ascii/hex representation of *file*. This will be a left-to-right hexadecimal representation of the bits in the *screendump* with each raster(row) being delimited by a newline character.

–r *file* is read and the standard output will be binary data in the following format:

2 bytes containing the number of rasters(rows).

2 bytes containing the number of bits/raster.

The actual bitmap itself.

This option was provided for the eventual introduction of printers other than **C.itoh**; for example, the UNIX system printer.

SEE ALSO

lpg(1), screendump(1).

NAME

cip – DMD interactive drawing system

SYNOPSIS

cip

DESCRIPTION

The *cip* utility is an interactive drawing system running under *layers* on the DMD terminal. Using a set of predefined *pic* [see *pic*(1) in the *UNIX System V DOCUMENTER'S WORKBENCH (DWB) Software Introduction and Reference Manual*] figures, the user can draw pictures within the *cip* picture frame. These pictures can be stored in a file and later formatted for output to a typesetter using *pic* or retrieved for editing or viewing.

SCREEN

The *cip* screen consists of three areas. The top-most area contains pictures of figures (templates). These templates are used to draw pictures in the next area of the screen. The middle area of the screen (picture) is used for drawing, storing, and editing pictures. The bottom of the screen is also broken up into three different areas. The left side displays the number of bytes of memory remaining in the DMD or error messages. The right side shows the current functions available when using the mouse buttons. Between these is a region used in conjunction with macros. When a macro is being edited, a box is drawn in this area showing the level of macro being modified.

MOUSE

The mouse is used for selection of all menu items, selection of templates, and editing of figures in the picture area. Button 1 is always used to select a template or a figure. Button 2 is used to edit a selected figure, and button 3 displays a menu which varies with the current state of *cip*.

Button 1 When button 1 is pressed in the template area, the rectangle containing the mouse arrow is displayed in reverse video. This indicates the currently selected template (Note: the function display at the bottom of the screen changes, too). The template can then be used to draw a figure in the picture area (see Button 2). If button 1 is pressed while in the picture

area, then the closest figure is selected for editing. The figure selected is blanked out as long as button 1 is pressed. If there is no figure on the screen or if the mouse cursor is far enough away from all figures, then *cip* returns to its original menu and any figure or template is unselected.

Button 2 Button 2 is only active when either a figure or a template has been selected. The functions available depend greatly on the figure selected. In general, when a template is selected, button 2 can only be used for drawing. However, if a figure is selected then button 2 is used for either copying, moving, or editing the figure.

Button 3 Button 3 has three uses. The button is used most often to display a menu. The menu shows different options available depending on the current state of *cip*. The button is also used to reshape the layer and to exit spline drawing when the spline template has been selected.

If the *cip* layer is not the full size of the screen then the button 3 menu will contain only two options. These options are for quitting **cip** or reshaping the layer to the full size of the screen.

If no figure or template has been selected, then button 3 allows the user to get and put files, clear the screen, redraw the screen, draw the screen in reverse video, place a grid on the screen, define a macro, display memory, and quit *cip*. The other menus displayed by button 3 depend on the currently selected figure.

DRAWING

After a template has been selected using button 1, button 2 is used to draw the figure within the picture area. The location and size of the drawn object are determined by the motions of the mouse and button 2. Each of the basic figures has a number of "sticky points". As the cursor is moved near these points, it will tend to stick to them until the cursor is moved sufficiently far

away. Figures selected and drawn as a single point are automatically erased.

Circles To draw a circle, select the circle template with button 1, move the cursor to the desired position for the center of the circle, and press button 2. As long as button 2 is pressed, the trial radius of the circle will track the mouse movement. When button 2 is released, the circle radius is fixed as the distance between the origin and cursor position.

Ellipses Ellipses are drawn in similar manner as circles.

Boxes Boxes are swept out with one corner of the box tracking the cursor and the other corner remaining stationary where button 2 was first pressed.

Lines Lines act similar to boxes with one end of the line fixed and the other end tracking the cursor.

Arcs Arcs are drawn by positioning the cursor to where the first point on the arc is to be located, pressing button 2, moving the cursor to the end point of the arc and releasing button 2. If the first point is to the left of the last point, then three quarters of an arc will be drawn; otherwise only one quarter of the arc is drawn.

Splines Splines are drawn with repetitive clicks of button 2. Each click adds a line segment to the framework of the spline. If button 2 is held down, the line will track the motion of the cursor. Only the framework of the spline is shown in the layer while the spline is being entered. To terminate and draw the spline, click button 3.

Text Text is entered in the layer by moving the cursor to the desired position for the center of the text and clicking button 2. The text is then typed in, terminated by a carriage return. The last character can be erased by using the backspace key. After the carriage return has been typed, the text will be displayed centered at the desired location.

COMMAND MENU

The command menu appears whenever no drawn figure is selected and button 3 is pressed. This menu contains commands that affect the whole screen or some part of it, but not a specific single figure. The following list describes the effect of selecting each command menu entry.

get file Read a *pic* file into *cip's* picture area. This file could have been created by editing a file or using *cip*. When this option is selected the user is prompted for the filename. When typing the name of the file, the backspace key can be used to delete the last letter and **control w** can be used to delete the last word. The name of the file is always terminated with a carriage return. After the name has been typed an outline of the figure being read is displayed in the layer. The user has the option to use either button 1 to center the figure on the screen or button 2 to move the figure. NOTE: Unlike previous versions of *cip*, the screen is not cleared before reading.

put file Store the current picture in a file in *pic(DWB)* format. This file can be used for future selection, editing, or printing. The name of the file is always terminated with a carriage return.

clear screen Deletes the picture currently being drawn. The user must verify this command by a second depression of button 3. If either buttons 1 or 2 are pressed, then the picture is not cleared.

redraw screen
Redraws all displayed figures. NOTE: Does not redraw cleared screen.

define macro
Allows a collection of figures to be treated as a single entity. The user must use button 2 to sweep out a rectangle around these figures.

grid Causes a grid of dots to be displayed in the layer. When the grid is turned on, figures being drawn will jump from grid point to grid point as the user moves

the cursor. The cursor favors sticky points; it will move to the nearest grid point only when there is no close sticky point. The grid is turned on and off by selecting this menu option.

reverse video

Flips the video sense of the layer. If the screen shows dark figures on a light background, it will be flipped to show light figures on dark. If the screen shows light figures on a dark background it will be flipped to show dark figures on light.

OBJECT MENUS

If a drawn figure is selected with button 1, then button 3 is pressed, a menu of entries that in some way transforms that selected figure is displayed. The contents of the menu are tailored to the class of figure selected. For example, instances of circles and ellipses can be copied or deleted. The following discusses each of these menus according to the figure selected.

Circle

delete The selected circle is deleted. No figure will be selected after the circle has been deleted.

copy The selected circle is copied when button 2 is pressed. The new circle will be drawn with its center located at the cursor position. Once the copy option has been selected, button 2 can be used repeatedly to make multiple copies. This option remains in effect until the current figure is unselected or the delete menu option is selected.

Box

delete The selected box is deleted.

copy The selected box is copied when button 2 is pressed. The upper left hand corner of the new box will be located at the position of the cursor when button 2 is pressed. This option can be repeated as long as the rectangle remains selected and no other option is used.

solid
dotted
dashed The outline of the box is made solid, dotted or dashed. This action is independent of the box's original outline and can be changed as many times as desired.

Ellipse

delete The selected ellipse is deleted.

copy The selected ellipse is copied when button 2 is pressed. The center of the new ellipse is located at the cursor position.

Line

delete The selected line is deleted.

copy The selected line is copied when button 2 is pressed. The origin of the line is always located at the cursor position. The origin is that point at which the original line was drawn. This is still true when a line is reflected.

solid
dotted
dashed The line is made solid, dotted or dashed.

arrow An arrow head is drawn on the end of the line nearest the cursor. This option acts as a toggle. If the line has no arrow then one is drawn and if the line has an arrow then it is undrawn.

reflect x
reflect y The line is reflected about either its x- or y-axis, respectively. The x-axis and y-axis are imaginary lines drawn through the center of the line.

Arc

delete The selected arc is deleted.

copy The selected arc is copied when button 2 is pressed. The starting point of the arc will be located at the cursor position when button 2 is pressed. The starting point of the arc is always the point at which the original arc was drawn from.

reflect x

reflect y The arc is reflected around either its x- or y-axis, respectively. The x-axis and y-axis are imaginary lines drawn through the center of the circle which includes the arc.

Spline

delete The selected spline is deleted.

copy The selected spline is copied when button 2 is pressed. The starting point of the new spline is determined by the starting point of the original spline and is located at the cursor position.

arrow The end of the spline closest to the cursor is drawn with an arrow. This option is a toggle, drawing an arrow if none exists and undrawing it if it does exist.

reflect x
reflect y The spline is reflected around either the x or y axis. The axes are imaginary lines drawn through the center of the minimum sized box which bounds the spline.

Text

delete The selected text is deleted.

copy The selected text is copied when button 2 is pressed. The new text will be drawn with its center located at the cursor position.

point size A second selection of button 3 will present a menu of point sizes. Selecting an item from the menu will change the text to that size.

roman
italic
bold The selected text is changed to the appropriate font style.

center
left justify
right justify The text is either centered, left or right justified in relation to the center of the text.

Macro

delete The selected macro is deleted.

copy The selected macro is copied when button 2 is pressed. The upper left hand corner of the new

macro will be located at the cursor position when button 2 is pressed.

reflect x
reflect y The selected macro is reflected around either the x or y axis. The axes are determined by the center of the macro.

edit The **edit** entry allows components of the macro to be modified, deleted, or added. Selection, drawing, editing and menus work exactly as described above except they operate within the context of the macro. When editing begins, a small box appears on the screen to the left of the **Mouse Buttons** box. It shows the current depth of macro editing. To terminate the editing, move the cursor inside the **Edit Depth** box and click button 1, decrementing the depth count. Changes made to the components will change all instances of the macro, except those that have been reflected.

separate Separate the macro back into individual figures.

EDITING

Figures that have been selected using button 1 can be modified using button 2. All the figures except splines, lines and arcs can be moved.

Circles/Ellipses

Once a circle/ellipse has been selected, it can be moved by positioning the cursor near its center and pressing button 2. The circle/ellipse will follow the cursor around the layer until the button is released. A circle/ellipse can be resized by pressing button 2 with the cursor close to the edge of the circle. Then as the cursor is moved, the circle/ellipse will either grow or shrink in size.

Boxes A box is moved by pressing button 2 while the cursor is close to the center of the box. The box then moves with the cursor until the button is released. If button 2 is pressed when the cursor is close to one of the corners, that corner will be fixed to the cursor

and move along with the cursor as it is moved.

Lines When button 2 is pressed, the end of the line nearest the cursor is repositioned to the location of the cursor. That end of the line will then move with the cursor as long as button 2 is pressed.

Arcs Arcs have three points which will track the cursor: the two end points and the center of the circle which contains the arc. The point nearest the cursor when button 2 is pressed is the one that will do the tracking. The center of the circle is shown by a pair of lines when the arc is selected.

Splines The framework of a spline is shown whenever the spline is selected. The points that make up this framework can be repositioned. The point nearest the cursor, when button 2 is pressed, will move with the cursor until button 2 is released.

Text Text can only be moved using button 2. When button 2 is pressed, the text is redrawn with its center aligned with the cursor. As the cursor is moved, so is the text.

Macros When button 2 is pressed, the figures within the macro are undrawn and a block appears showing the outline of the macro. This box will track the cursor. When button 2 is released, the box disappears and the figures within the macro are redrawn in their new location.

KEYBOARD

The keyboard is only active for particular menu selections and is used only for typing file names and typing text. A backspace can be used to erase the last typed character. Entering **control W** will erase the last word. Entering **control U** will erase the entire line. The function keys are also available for entering text.

OTHER PROGRAMS

pic(1) generates typesetter commands for *cip* pictures

proof(1) program to preview pictures drawn by *pic*(1)

FILES

$DMD/lib/cip.m layers version of *cip*
$DMD/bin/cip shell script for executing *cip*

SEE ALSO

proof(1).
pic(1) in the *UNIX System V DOCUMENTER'S WORKBENCH Software Introduction and Reference Manual.*

WARNINGS

A picture stored in a file will **not** be read correctly by *cip* if it contains a definition of a picture which is too large for the *cip* picture frame.

The *cip* utility relies on the host machine only for file transfers. If the host crashes while you are creating or modifying a picture, there is no way to save the picture. *Cip* will maintain all requested fonts on the terminal, until *cip* is exited. This is a feature which prevents unnecessary downloading of fonts. The user should be aware that once this memory is used it will not be released. On terminals with small amounts of available memory, this problem can be avoided by planning the picture before requesting the fonts. The macro facility is intended to work with small objects. Even though large macros can be created and written, they cannot be handled by *pic*.

BUGS

Cip does not know the difference between a *pic* file and a non-pic file. If an attempt is made to read a file such as a directory or text file, *cip* will hang.

NAME

demo - demonstrations available on the DMD

SYNOPSIS

demo [options]

DESCRIPTION

The *demos* are graphical programs that run on the DMD. They can run in stand-alone, layers, or both environments. Only one demo can be run in a layer or on the screen (stand-alone) at a time. Typing 'q' will exit the program. Typing *demo* with no options gives you a list of demonstrations available in your current environment (stand-alone or layers).

a (B) interesting representation which demonstrates some graphical capabilities of the DMD.

b (B) ricocheting squares.

ball (B) reverse video of nball.

bltdemo (S) an interesting graphical display.

bounce (B) a graphical representation of a bouncing line.

calc (L) a desk top calculator.

clock (L) displays face of clock with moving hands in addition to a digital time display.

clock1 (L) same as clock but with larger hands.

clock2 (L) a clock with no personality

disc (S) draws a circle which gets filled in by smaller circles. This is a fast demo that exits when it is done.

dmdlock (L) a useful demo thats locks up the keyboard and mouse by the use of a password.

doodle (B) use "mouse" for doodling. Button 1 draws and button 2 clears the screen.

elevator (S) an interesting 3D demonstration.

juggle (L) balls being juggled. Hit '+' to add balls '-' to delete balls, 'h' to increase arc, and 'l' to lower arc.

life (L) a demo of the algorithm of life.

m (B) messy squares. Use space bar to reset.

maxwell (L) graphical representation of Maxwell's distribution of molecular energy principle. Button 2 of "mouse" opens a door, allowing molecules to exchange sides.

moire (S) an interesting graphical display.

munch (S) munching squares.

nball (B) a bouncing ball.

rae (B) balls falling into an empty box until it is filled.

smiley (S) game where 'frowny' tries to nuke-bomb the 'smileys' out of the sky. Left arrow (<-) moves 'frowny' left and right arrow (->) moves 'frowny' right. The space bar fires the nuke bombs. The game will reset itself and pressing space bar starts new game. A challenging game for kids of all ages.

spheres (S) fills the screen with spheres.

weird (B) flying balls.

win (S) displays three windows and allows the user to create windows by depressing button 3 on the " mouse" and sweep across any diagonal.

memory (L) shows the amount of memory available for different usages.

rose (L) rotates trigonomic figures to produce a flower shape (rose).

scope (L) a demo that monitors the user's processes on a oscilloscope type screen.

worm (L) an arcade type game, which a worm grows as it eats the bugs on the screen. The speed levels can be changed with the three vertical indicators on the right of the demo.

NAME

dmdar – archive and library maintainer for portable archives

SYNOPSIS

dmdar key [posname] afile [name] ...

DESCRIPTION

The *dmdar* command maintains groups of files combined into a single archive file. Its main use is to create and update library files as used by the link editor. It can be used, though, for any similar purpose. The magic string and the file headers used by *dmdar* consist of printable ASCII characters. If an archive is composed of printable files, the entire archive is printable.

When *dmdar* creates an archive, it creates headers in a format that is portable across all machines. The portable archive format and structure are described in detail in *dmdar*(4). The archive symbol table (described in *dmdar*(4)) is used by the link editor *(*m32ld*(1))* to effect multiple passes over libraries of object files in an efficient manner. An archive symbol table is only created and maintained by *dmdar* when there is at least one object file in the archive. The archive symbol table is in a specially named file which is always the first file in the archive. This file is never mentioned or accessible to the user. Whenever the *dmdar*(1) command is used to create or update the contents of such an archive, the symbol table is rebuilt. The **s** option described below will force the symbol table to be rebuilt.

Key is an optional **–**, followed by one character from the set **drqtpmx**, optionally concatenated with one or more of **vuaibcls**. *Afile* is the archive file. The *names* are constituent files in the archive file. The meanings of the *key* characters are:

d Delete the named files from the archive file.

r Replace the named files in the archive file. If the optional character **u** is used with **r**, then only those files with dates of modification later than the archive files are replaced. If an optional positioning character from the set **abi** is used, then the *posname* argument must be present and specifies that new files are to be placed after (**a**) or before (**b** or **i**) *posname*. Otherwise new files are placed at the end.

q Quickly append the named files to the end of the archive file. Optional positioning characters are invalid. The command does not check whether the added members are already in the archive, and is useful only to avoid quadratic behavior when creating a large archive piece-by-piece.

t Print a table of contents of the archive file. If no names are given, all files in the archive are tabled. If names are given, only those files are tabled.

p Print the named files in the archive.

m Move the named files to the end of the archive. If a positioning character is present, then the *posname* argument must be present and, as in **r**, specifies where the files are to be moved.

x Extract the named files. If no names are given, all files in the archive are extracted. In neither case does **x** alter the archive file.

v Give a verbose file-by-file description of the making of a new archive file from the old archive and the constituent files. When used with **t**, give a long listing of all information about the files. When used with **x**, precede each file with a name.

c Suppress the message that is produced by default when *afile* is created.

l Place temporary files in the local current working directory, rather than in the directory specified by the environment variable **TMPDIR** or in the default directory **/tmp**.

s Force the regeneration of the archive symbol table even if *dmdar*(1) is not invoked with a command which will modify the archive contents. This command is useful to restore the archive symbol table after the *m32strip*(1) command has been used on the archive.

FILES

/tmp/dmdar* temporaries

SEE ALSO

m32ld(1), m32lorder(1), m32strip(1), tmpnam(3S), dmda.out(4),

dmdar(4).

BUGS

If the same file is mentioned twice in an argument list, it may be put in the archive twice.

NAME

dmdcat - send files to a printer; used in conjunction with *dmdp(1)*.

SYNOPSIS

dmdcat [**–b**] [file...]

DESCRIPTION

The *dmdcat* command is intended to be used in conjunction with *dmdp(1)*, in order to send files to a printer. This command will send data to a 5620 terminal preceded by a *Printer On* escape sequence and succeeded by a *Printer Off* escape sequence. The escape sequences sent are:

Printer On - ESC[?4i

Printer Off - ESC[?9i

Dmdcat(1) is used in conjunction with the *host* option of *dmdp*. *Dmdcat* will print the concatenation of files specified on its command line, or its standard input if no files are specified.

Dmdcat has one option **–b**, which will strip backspaces from the output of *dmdcat*. If backspaces would result in two or more characters appearing in the same place, only the last character read will be output. This means that the printed output will appear exactly as it appears on the 5620 screen, without bold and underline. This option is useful for printers which either cannot process backspaces or are slow in processing backspaces.

To use *dmdcat* in conjunction with *dmdp*, refer to "The Host Option" section of *dmdp*(1). First, *dmdp* must be running in another layer. Select the host option of the *dmdp* menu and point to this layer. Now, *dmdp* will capture the output from *dmdcat* and send the escape sequence to turn the printer on, *cat* the file to the printer, and turn the printer off.

DIAGNOSTICS

Dmdcat is a shell program which uses the *col*(1) command to strip backspaces with the –b option. Col is not available on all UNIX systems. The –b option will give an error message if it cannot locate the *col* command.

SEE ALSO

dmdp(1).

cat(1), col(1) in the *UNIX System V User Reference Manual.*

NAME

dmdcc – DMD C compiler

SYNOPSIS

dmdcc [**–J**] [**–c**] [**–g**] [**–O**] [**–OA**] [**–W**c,arg1[,arg2...]] [**–S**] [**–P**] [**–E**] [**–V**] [**–D**symbol]... [**–U**symbol]... [**–I**dir] file ...

DESCRIPTION

The *dmdcc* command is the DMD C compiler. Any software developed to run in the DMD must be compiled using this command. It works in a very similar manner to other cc commands (in particular the *m32cc* command), though it also calls the link editor with certain arguments to ensure correct operation of code on the compiler.

The *dmdcc* utility generates IS25 assembly instructions. It accepts three types of arguments:

.c
.s
.o

Arguments whose names end with **.c** are taken to be C source programs, and those with **.s** are taken as assembly programs. They are compiled and assembled each object program is left on the file whose name is the source with **.o** substituted for **.c** or **.s**. The **.o** file is normally deleted, however, if a single C program is compiled and link-edited all at one go.

The following flags are interpreted by *dmdcc*. See *m32ld*(1) or *m32as*(1) for other useful flags.

–J Compile the named files, loading and linking them for standalone execution on a DMD. If this flag is not given, the files will be loaded and linked for *layers* execution; that is, execution within a layer.

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

–g Flag to the compiler to produce additional information needed for the use of *dmdebug*(1).

–O Invoke an object-code optimizer. The optimizer will move, merge, and delete code; so symbolic debugging with line numbers could be confusing when the optimizer is used.

–OA Optimize assembly. Invoke the object code optimizer for a **.c** source file which contains assembler escapes. Use this flag at your own risk. The optimizer does not recognize all .PP assembly language instructions.

–W*c*,arg1[,arg2...]
Hand off the argument[s] to pass *c* where *c* is one of [**p012al**] indicating preprocessor, compiler first pass, compiler second pass, optimizer, assembler, or link editor, respectively. For example:

–Wa,–m

invokes the *m4* macro preprocessor on the input to the assembler.

–S Compile the named C programs, and leave the assembler-language output on corresponding files suffixed **.s**.

–P Run only the macro preprocessor on the named C programs, and leave the output on corresponding files suffixed **.i**.

–E Same as the **–P** option except the output is directed to the standard output. This allows the preprocessor to be used as a filter for any other compiler.

–V Print the version of the DMD compiler, optimizer, assembler, or link-editor that is invoked.

–D Define *symbol* to the preprocessor. This mechanism is useful with the conditional statements in the preprocessor by allowing symbols to be defined external to the source file.

–U Undefine *symbol* to the preprocessor.

–I Change the algorithm for searching for #**include** files whose names do not begin with / to look in *dir* before looking in the directories on the standard list. Thus, #**include** files whose names are enclosed in "" will be searched for first in the directory of the *file* argument, then in directories named in **-I** options, and last in directories on a standard list. For #**include** files whose names are enclosed in <>, the

directory of the *file* argument is not searched.

Other arguments are taken to be either link-editor flag arguments, or C-compatible object programs, typically produced by an earlier *dmdcc* run, or perhaps libraries of C-compatible routines. These programs, together with the results of any compilations specified, are link-edited (in the order given) to produce an executable program with name **dmda.out** unless the **–o** option of the link-editor is used.

The *dmdcc* command expects two shell variables to be set and exported in the user's environment. This should be done by your system administrator. The variable **DMD** must point to the "root" directory of the DMD software node. The variable **DMDSGS** must point to the "root" directory of the remainder of the SGS (that is, the compiler, loader, and assembler).

FILES

file.c	input file
file.o	object file
file.s	assembly language file
dmda.out	link-edited output
/usr/tmp/dmd?	temporary
/lib/cpp	preprocessor
LIBDIR/comp	compiler
LIBDIR/optim	optimizer
LIBDIR/libc.a	DMD library

SEE ALSO

dmdebug(1), m32as(1), m32cc(1), m32dis(1), m32ld(1), m32list(1).
m4(1) in the *UNIX System V User Reference Manual.*

Kernighan, B. W., and Ritchie, D. M., *The C Programming Language*, Prentice-Hall, 1978.
Kernighan, B. W., *Programming in C–A Tutorial.*
Ritchie, D. M., *C Reference Manual.*

DIAGNOSTICS

The diagnostics produced by the C compiler are sometimes cryptic. Occasional messages may be produced by the assembler or link-editor.

NAME

dmdck – DMD executable file auditor

SYNOPSIS

dmdck

DESCRIPTION

The *dmdck* utility audits the existence and mode of the Core Package, Text and Graphics, Application Package, and Application Development Package for the DMD Terminal Software.

If the location of the DMD Software ($DMD) is known, change directories to $DMD and type:

```
dmdck <cr>
```

where <cr> is a carriage return. This will begin the audit. Diagnostic messages will be printed to the screen with **bell** tones (control g) for each unexpected audit result. A summary of all unexpected audit results will be printed at the finish of the audit and saved in the file *dmdck.errors* in the same directory as *dmdck*.

If the location of the DMD root node is unknown and you wish to find it, move to the " /" directory and type:

```
find . -name "dmdck" -exec {} ; <cr>
```

If a prompt is returned with no other output, *dmdck* is not on the subject machine.

The *dmdck* utility first *makes* the executable *file* for the host machine. This *make* is done to avoid object code incompatibility problems.

The *dmdck* utility does **not** make any DMD software.

FILES

$DMD/file.c
$DMD/file.mk

SEE ALSO

make(1) in the *UNIX System V User Reference Manual.*

NAME

dmdebug – symbolic debugger

SYNOPSIS

dmdebug [**–u** file] [**–m** layersys] [**–t** dmdebug.m] [**–l** 32ld]

DESCRIPTION

Dmdebug is a source-level symbolic debugger for C programs running under *layers*. Running asynchronously and communicating by means of the keyboard and mouse in its own layer, *dmdebug* provides for controlled execution of another process running under *layers*. It takes advantage of the source symbol tables generated by the **–g** option of *dmdcc*.

The options are:

–u *file*	used to specify user program symbol tables
–m *layersys*	used to specify alternate layersys symbol tables
–t *dmdebug.m*	used to specify alternate *dmdebug* downloaded portion
–l *loader*	used to specify an alternate loader to download *dmdebug.m*.

MODEL

Most commands manipulate and are interpreted with respect to three major values:

i) *state*: the state of the debugged process

ii) *current context*: the active statement in one of the functions on the call stack

iii) *current location*: an arbitrary address within DMD memory.

If there is a state, it is displayed in a status line at the top of *dmdebug*'s layer; otherwise an appropriate message appears there.

The subject state variable may have one of the following values:

RUNNING

HALTED

BREAKPOINT_TRAPPED SINGLE_STEP_TRAPPED

EXCEPTION_TRAPPED

The RUNNING state and the various TRAPPED states have obvious interpretation. A process is HALTED when a flag is set in its process descriptor to prevent layersys from scheduling it to run. If there is a current context, the last item on the button 3 menu is '*func_id*() vars', where *func_id* is the function. If the current location is non-zero, it appears as the normal command prompt.

INTERACTION

Apart from the status line (which video inverts momentarily whenever its contents change), *dmdebug*'s layer scrolls conventionally.

The *dmdebug* utility usually prompts ':' (or '*address*:' with the current location) when it is ready to accept a command entered from either the keyboard or a mouse menu. If a 'dead mouse' icon replaces the cursor, the mouse may not be used. This happens only when prompting for the symbol table file. The 'menu' icon that replaces the cursor shows when there are menus available on buttons 2 and/or 3. It is always safe to raise a menu. If it is not what you want, do not select anything and there is no harm done. When *dmdebug* is busy for a long period, a 'coffee cup' icon replaces the cursor.

If invoked with **–m**, *dmdebug* uses the specified *file* to obtain symbol table information for *layers* itself (the system standard is default). The **–u** flag specifies a source for symbols for the layer being debugged (see *layer* below).

MOUSE

Buttons 2 and 3 display command menus. Button 3 commands treat the process only at the source language level. Button 2 commands allow further access to the terminal's raw memory. Some commands call for confirmation by a second button hit on a 'skull' cursor.

MENUS

Selecting nothing from a menu is always safe; it has no effect. Selecting 'more...' at the top or bottom of a menu, moves to another portion of a menu which is too large to be displayed. Menus too large to fit into the fixed structures in the terminal are

broken into segments that will fit, with a special item at the top and/or bottom to move to the adjoining segments: The standard menu primitive in the terminal process was modified to scroll quickly through menus that are too large to be popped up, using a "scroll bar" beside the menu to control which 12 items are visible. Menus are broken into segments of 64 items, which can be scrolled with 12 items visible at any one time.

Button 3

layer Selects a new layer for examination. The layer is identified by hitting the layer with the 'bullseye' cursor. The layer is flashed by video invert. The name of the object file is displayed in the **dmdebug**

window as: argu[o]=<filename> symbol table?

Where filename is the name of the object file with **-g** symbol tables. Button 3 must then be used to select one of the following options:

argu[o] keyboard none

The given filename is used if " argu[o]" is chosen. A different file can be specified from the keyboard if " keyboard" is chosen. If none of the items or " none" is chosen, then **dmdebug** will use no symbol tables. The default file name is usually taken from the *32ld* command which loaded the layer. It is, therefore, best to give an *absolute* path name when using *32ld* and *jx* in shell scripts and programs. This default may be changed at the time of *dmdebug* invocation by using the **–u** command line argument.

quit Must be confirmed.

breakpts Repeatedly offers a further menu on button 3 with the commands **list** [*n*]**, clear all,** and a list of functions in the program. If no breakpoints are currently set, the list and clear entries are

replaced by an entry indication '<none set>'. The value in brackets with **list** shows how many breakpoints are currently set. The commands list and clear them all, respectively. **Clear all** must be confirmed. Selecting a function identifier offers a terminal menu with:

call
return
both
none

One of these is tagged by '>', indicating what breakpoints are set. Selecting one modifies the breakpoints on the function. The breakpoint at function call is *after* the stack frame has been advanced, but *before* initialization of the registers and automatics. The breakpoint on function return is *after* the return value has been set, but *before* the stack frame is retracted. It is safe to set and clear breakpoints *while the process is running*. To set breakpoints before the program starts, use the –z option on *32ld* to leave the process halted after loading. Breakpoints can only be set on individual C statements or physical locations by means of the keyboard (below).

globals Repeatedly offers a menu with the program's globals. Each one selected is displayed.

stmt step Executes a single statement of the program. After it is executed, the status line shows the next statement to be executed. It does not set the current context, however; **function** or **traceback** must be used explicitly before local variables are accessible. Statement stepping can only be used after stopping on breakpoint. It is not available when the process is merely suspended by **halt**.

go Resumes execution of the process.

halt Suspends execution of the process.

traceback Lists the active function stack frames from the one that is executing back to main(). Each line in the traceback has the form:

file:*line*[+*offset*] in *function*(*arg*, ... *arg*)

giving the statement being executed as source file, line number (and instruction byte offset) within a function and its arguments. The deepest stack frame becomes the current context.

function Offers a menu of functions (in traceback order) from which one may be selected as current context.

func_id() **vars** Repeatedly offers a menu of the local variables, arguments and statics which are alive at the active statement in the current stack frame. Each variable is classified as 'lcl' (local), 'reg' (register), 'sta' (static) or 'arg' (argument).

EXPRESSION EVALUATION

Dmdebug recognizes and evaluates a subset of C expressions that includes fixed point arithmetic, subscripting, indirection, field selection, assignment and function application. Expressions may be entered from the keyboard. In fact, most expressions evaluated are those of a further subset that are built from menu-driven interaction. Pointers and aggregate variables can be used as the basis for building expressions in which the user is repeatedly offered menus of applicable operators and functions. This begins with the selection of a variable from a menu, such as the variables in a function with an argument, *pt*, and a local variable, *i* :

pt arg **i** **lcl** Suppose **pt** is selected, a pointer to a record of type **Point** :

```
struct    Point {
        short     x;
        short     y;
};
```

```
struct      Point      *pt;
```

The expression is evaluated. Its type determines which operators are applicable, and they are presented as a menu of expressions in which the expression-in-hand, **Point**, is represented by a tilde:

~[?]
~->x
'->y

Selecting ~->**x** or ~->**y** causes the function to be re-invoked recursively with the expression **pt->x** or **pt->y**, respectively. Both of these are the simple case where the expression evaluates to a scalar; the value is printed and the function returns to the level above, where the same menu is presented again. Selecting ~[**?**] produces a scrolling menu of subscripts (with a somewhat arbitrary range of -4..128):

-4
-3
-2
-1
0
1
2
3
4
5
6
.
.
.

Selecting one of these subscripts re-invokes the function with a new expression like **pt[2]**, in which case the expression is a record, not a pointer, and the selections in the next menu are:

~.x
~.y

where the tilde now represents **pt**[**2**] .

In fact, the menu above also contains special entries which are not normal expressions:

Point{~}
~.x
~.y
%point(~)

Point{~} prints the whole record, say: {x=226,y=413} **%point(~)** calls a graphics function in the terminal to display a cross-hair at that point on the screen. **%point(~)** is included in the menu because *dmdebug* knows that struct Point is one of the primitive graphics data structures defined for all DMD programs. Other functions display **Rectangle**s, **Texture**s, **Texture16**s and **Bitmap**s. "Off-screen" bitmaps are a good example; the programmer can see the effects of graphics operations on images as they are constructed, before they are copied to the region of memory that is mapped to the display.

This general mechanism supports chasing through arrays and through arbitrary linked data structures. In addition to the built-in operations at each step, monadic functions in the program, whose argument is the type of the expression-in-hand, are sought from the symbol table and included in the menu. Again, it is the responsibility of the user to know when it is safe to invoke a function.

Pointer Chasing

If a variable selected from the global or local menu is a structure, a pointer to a structure or an array, it is displayed as one of:

variable .	<field>
variable –>	<field>
variable	<index>

respectively. The menu on button 3 can then be used to complete the expression and display its value. While the expression this yields is still an aggregate, a further selection must be made to extend it. Making no selection or hitting button 1 or 2 'backs up' out of an expression.

Button 2

Button 2 has commands which root around in memory displaying it as bytes, shorts (16 bits), long words (32 bits), structures, unions, and enumerations. Before using this, the command

. expression

is needed to set the current location to the value of the expression.

The model for this button is that the menu itself is a window into memory. Selections from the menu change the location, size, and format of the objects seen the next time it is raised and can copy useful ones into the scrolling area.

The top three menu items are the location and value for the current memory location (tagged by '.') and its two neighbors. Selecting one of these three writes it into the scrolling area and makes that location the current one.

Selecting **byte, short,** or **long** fixes the size of objects viewed. Selecting **decimal, octal, hex,** or **ascii** fixes the radix. The ascii radix displays each byte as a C character constant (e.g., 'a', '\n', '\277').

For pointer chasing, if the object size is **long** and the value at the current location makes sense as an address, the location referenced and its value are shown (tagged by '*'.) It can be picked to become the current location.

String shows the zero-terminated strings starting at the current location, and at current–4 and current +4. Long strings are truncated.

Struct and **enum** present menus of the structures and enumerations in the program. Selecting one interprets memory as a vector of such objects. Most structures are horribly truncated in the menu, but are displayed in full when selected and displayed in the scrolling area.

If there is a symbolic form for the current location it is displayed as an item tagged by '.'. Selecting this gives a fuller symbolic form.

A final item may appear in this menu. If *dmdebug* is unable to display an object when given its symbolic name, it saves the address and places it in the menu. It may be picked to become the current location.

KEYBOARD

Dmdebug repeatedly prompts with the address of the current memory location and accepts one or more of the following commands separated by semi-colons. Constants may be unsigned decimal, octal, or hexadecimal, in C notation.

.la	Select a new layer to debug.
.exit	Exits.
^D	Exits.
expression	Evaluate and print its value. The following operators are supported: + – * / % –> [ ] . = (binary) & * – (unary) Functions may also be called, but are executed by *dmdebug*'s process (while the user's is suspended if necessary.) Structures may be expressed as *struct_id(expr, ... expr)*, for example, Point(100,200). Identifiers are bound to variables as they would be in the current context using C's scope rules.
.v	Print all local variables visible from the current context.
.t	Display a function traceback. It sets the current context to the deepest user function.
.h	Halt the process.
.g	Start the process.
.?	Display the current state of the debugged process.
.bc *function_id*	Set a breakpoint at function call.
.br *function_id*	Set a breakpoint at function return.

.bs *file line_no*	Set a breakpoint at a given line within a source file.
.cc *function_id*	Clear breakpoint at function call.
.cr *function_id*	Clear breakpoint at function return.
.cs *file line_no*	Clear breakpoint at a given line within a source file.
.bl	List all breakpoints.
.ca	Clear all breakpoints.
.c [*count*]	Set the new current context to the function which the present one calls. When given, iterate the number of times specified by the constant. This and the following command are used in conjunction with the main menu entry *function*, which permits one to examine previous stack frames in the call sequence.
.r [*count*]	Set the new current context to the function to which the present one returns. When given, iterate the number of times specified by the constant.
. *expression*	Set the address of current memory location to the value of the expression.
[±] [*offset*]	Set the current location relative to itself by the given constant times the size of the object last displayed. The null command (a carriage return) is equivalent to +1.
.x	Find a symbolic name for the current memory location.
.b	The current location is displayed as a byte.
.sh	The current location is displayed as a short.
.l	The current location is displayed as a long.
structure_id	The name of a defined structure displays memory from the current location as an instance of the structure. This also works for unions and enumerations.

%*register*	Set the current location to where the specified register in the current context was saved and display as a long (32 bits). *Register* is r[0–8], sp, fp, ap, psw, sp, pcbp, isp, and pc.
*	Set the address of the current location to the 32 bit address at the current location.
.li *file line_no*	Display the first instruction of code generated for the statement beginning at a line number of a source file.
.= *expression*	Write value of expression to the current memory location. Operand length is that of the previous .b, .sh, or .l command (or button 2 equivalent).
< *file*	Read command script from a file.
.bi	Set breakpoint on the instruction at the current memory location.
.ci	Clear breakpoint on the instruction at the current memory location
.si [*count*]	Single step the next instruction or number of instructions. This command can only be invoked if the layer being debugged is stopped at a breakpoint or prior single step.
.ss [*count*]	Single step the next C statement or number of statements. This command can only be invoked if the layer being debugged is stopped at a breakpoint or prior single step.
.*base*	Set radix for display of addresses, where *base* is 8, 10, or 16.
.cd [*directory*]	Change directory.
.pwd	Print the current directory.
.sd *directory*	Adds *directory* to the list of directories to search for source files.
.zP	Sets the current location to the address of the debugged process's process control block.

.zd *addr duration* Displays the terminal's raw memory on the screen for the given duration.

SEE ALSO

dmdcc(1), layers(1), 32ld(1).
sdb(1) in the *UNIX System V User Reference Manual.*

DIAGNOSTICS

The diagnostics produced by *dmdebug* are intended to be self-explanatory. Messages prefaced by '5620: ' are generated and printed autonomously by the terminal program when the state of the debugged process is not as expected. They appear in exceptional situations where it is hard for *dmdebug* to keep track of events. Examples are: manually patching breakpoint traps, arbitrary stores into code or control memory, reloading of the debugged layer. Once such a diagnostic is given, there are no guarantees.

BUGS

The *dmdebug* program is still evolving and has some rough edges. The .= command can write anywhere in memory. Anonymous structures, unions and enumerations are not supported. For example:

```
typedef { int field } Record;
```

introduces an anonymous structure. A simple remedy is to name the structure thus:

```
typedef Record { int field } Record;
```

NAME

dmdp – printer support for 5620 DMD terminals

SYNOPSIS

dmdp printer_type

DESCRIPTION

The *dmdp* utility is a printer program which allows the user to send screen and host data from a 5620 DMD to a locally attached printer. The *dmdp* utility runs only under *layers*(1), and when the program is completely downloaded, a picture of a printer will appear.

In order to view the messages at the bottom of the *dmdp* layer, it is recommended that the layer must be at least one-quarter of the screen wide.

printer_type The printer_type argument specifies the type of printer attached to the 5620 DMD. Currently supported printers are:

- **5310** or **5320** for the Teletype 5310 or Teletype 5320 printers, respectively.

8510B for the C. Itoh 8510b printer.

thinkjet for the Hewlett-Packard HP 2225 ThinkJet printer.

transparent for transparently sending input from the host, through the 5620, to any printer or other RS-232 device. When this option is used, the **Screen** option (below) will not be available. Input is transparently passed through the 5620 to the RS-232 device with no processing of tabs, or insertion or monitoring of escape sequences. The host must insert delays, if necessary, to compensate for lack of flow control. The RS-232 device must be set to eight bits per character plus even parity.

The printer should be connected to the send only port (Port B) on the back of the 5620 DMD. Port B should be set to 9600 baud through terminal setup. It will be necessary to option your printer before using *dmdp* for the first time. Printer options for the

supported printers are listed in the 5620 Dot-Mapped Display User Guide.

The *dmdp* utility is a menu-driven program. There are two general types of menus: **action** menus accessible from button 3, and **setup** menus accessible from button 2. Action menus are used to initialize the 5620 for printing and to start and stop printing. Setup menus are used to set optional characteristics of the printer and the printed output.

The button 2 setup menu is only available before the printer is turned on. This is before **Print** is chosen in **Screen** mode or **Printer On** is chosen in **Host** mode. The button 2 setup menu is not available if the printer type is **Transparent**.

MOUSE BUTTON 2 SETUP MENU

The setup menu is used to set optional characteristics of the printer and the printed output. It also provides tools for sizing graphical pictures to fit onto a printed page. The exact contents of the button 2 setup menu will vary depending upon the printer specified on the *dmdp* command line, since some of the options are printer specific. When a setup item consists of the choosing between a number of options, the currently selected option will be pointed to by an arrow **(>)**. Each option in the Setup menu is marked by an **(S)**, **(H)** or **(SH)**, which specifies which printing mode(s) it applies to: screen, host, or both, respectively. A different option can be selected by moving the mouse cursor to the desired option and releasing button 2. The options are remembered for the current *dmdp* session, until *dmdp* is exited.

Show Printer Page This option shows what part of the 5620 screen will fit onto one printed page, both in width and length, by forming an outline on the screen. This outline can be moved with the mouse and will disappear when any mouse button is clicked.

Reshape to Page This option reshapes a layer's window to the largest possible size that can fit on one printed page. When this option is chosen, the mouse cursor will be changed to a target. The target should be positioned within the

layer to be reshaped, and then button 3 should be clicked.

Center The printed output will be centered in both the vertical and horizontal directions on the printed page. In order for vertical centering to work properly, the paper in the printer must be formfed before printing is started. In order for horizontal centering to work properly, the paper in the printer must be centered properly.

No Center The printed output will start at the left hand border of the page at the current position of the print head. This may yield faster printing and is intended for draft copies.

Off-screen Copy *Dmdp* will attempt to copy the screen area to be printed into off-screen memory before printing is started. If the copy succeeds, all terminal operations are unlocked and everything can proceed as normal, including modification of the area being printed, while printing is occurring. The area to be printed, highlighted in reverse video, will be changed to normal video before printing starts.

No Off-screen Copy No off-screen copy will be attempted, and the progress of printing can be monitored by watching inverse video turn into normal video on the 5620 screen.

Density This allows a user to change both the horizontal and vertical densities (Dots Per Inch) for the printer from the default value.

Print Quality This option is used to set printer specific print qualities, usually Draft Quality and Final Quality. In general, the tradeoff is print quality as compared to print speed.

Paper Width This option is available only if the printer specified on the command line can use paper

of different widths. The two options are " Wide Paper " and " Narrow Paper."

MOUSE BUTTON 3 ACTION MENUS

The main menu is displayed the first time button 3 is pressed after *dmdp* is downloaded, and is the menu that is always returned to when printing is completed. The main menu can also be returned to from any sub-menu by choosing the **Main Menu** item.

Main Menu

Screen prints a portion of the 5620 screen.

Host prints incoming traffic from the host computer. It is usually used to print text.

Exit exits the *dmdp* program.

Screen Option

First Level Sub-menus under the Screen Option

When **Screen** is chosen in the main menu, a sub-menu is available with button 3. This menu allows the user to pick the area of the screen to be printed. When an area is picked, a number of actions occur: The area that **will actually be printed** becomes highlighted in inverse video on the 5620 screen. Note that the area to be printed can be smaller than the area picked because of differences in print density between the 5620 screen and the printer, as discussed in section 2. The *dmdp* layer becomes locked in the terminal and cannot be deleted until control is returned to either the main menu or this first level menu under the screen option. If an attempt is made to delete the *dmdp* layer when it is locked in the terminal, the layer will momentarily flash. If the area to be printed is contained within a single layer, all activity in that layer will be halted. If the area to be printed is not contained within a single layer, all activity on the 5620 screen will be halted, and the entire terminal will be dedicated to *dmdp*.

Select Layer allows a layer to be selected as the target area. The mouse cursor (now a target) should be positioned within the layer to be selected, and then button 3 should be clicked.

Sweep Rectangle

allows any screen rectangle to be selected as the

target area. The method used to sweep the rectangle is identical to that of creating a new layers window.

Whole Screen selects the entire screen as the target area.

Main Menu returns *dmdp* to the main menu.

Second Level Sub-menus under the Screen Option

Print This will cause printing to start. As the area is printed, slices of the highlighted print area will be changed to normal.

Main Menu return *dmdp* to the main menu.

Third Level Sub-menus under the Screen Option while Print is On

This menu is available while *print* is on and before printing is completed.

Pause causes printing to pause.

Continue causes printing to continue.

Main Menu returns *dmdp* to the main menu. Note that this also has the effect of aborting printing.

The Host Option

When the **Host** option of the main menu is selected, the mouse cursor (now a target), should be positioned within the layer where input is to be printed, and then button 3 should be clicked. When a layer is selected in this manner, the *dmdp* layer becomes locked in the terminal and cannot be deleted until control is returned to the main menu.

There are three ways printing can be started: The first is to send an escape sequence from the host to the chosen layer. This is useful for printing without having print commands and prompts appear in the output. The escape sequences to turn the printer on and off are:

Printer On - ESC[?4i
Printer Off - ESC[?9i

These can be supplied by the *dmdcat*(1) program.

The second way to start printing is with the mouse, as explained below.

The third way is using *dmdcat* in another layer; *dmdcat* can turn on the printer, *cat* a file, and turn off the printer. [See *dmdcat*(1).]

First Level Sub-menus under the Host Option

Printer On will cause all input to the chosen layer to also be sent to the printer.

Main Menu returns *dmdp* to the main menu.

Second Level Sub-menus under the Host Option while Print is On

Printer Off will stop input to the chosen layer from being sent to the printer. Control will be returned to the previous sub-menu, and the layer chosen when **Host** mode was first entered will remain chosen.

Main Menu returns *dmdp* to the main menu. It will also stop input to the chosen layer from being sent to the printer.

FILES

$DMD/lib/dmdp.m	executable of *dmdp*
$DMD/bin/dmdp	shell script for executing *dmdp*

SEE ALSO

dmdcat(1), layers(1).
5620 Dot-Mapped Display User Guide
Release 2.0 cat(1) in the *UNIX System V User Reference Manual.*

NAME

hp2621 – Hewlett-Packard 2621 terminal emulator.

SYNOPSIS

hp2621 [**startup** [**first-time**]]

DESCRIPTION

The *hp2621* utility is an interactive terminal emulation program that runs under *layers*(7) on a DMD 5620 terminal. It allows the user to run programs that normally send/receive characters to/from the Hewlett-Packard 2621 terminal. In addition, it provides useful features such as screen memory and reverse video.

The variable $TERM should be set to **2621** and $TERM should be exported in order to use *hp2621*.

Startup is the command sent to the host every time a new *hp2621* layer is created. The maximum length of *startup* is 99 characters. *First-time* is the command sent to the host only when the first layer is created. Both commands get echoed in the layer which sent them. To exit *hp2621*, simply execute *jterm* in the *hp2621* window.

Character Input from Host

The *hp2621* utility interprets characters coming from the host and performs the appropriate *hp2621* function. Escape sequences and control characters (octal values less than 040) get interpreted in such a way that they perform a similar function on the DMD layer as they do on the *hp2621* terminal. The characters with octal values 040 - 176 are treated as normal characters; each character received is displayed at the current cursor position and the cursor is moved one character position to the right. If the cursor was in the right-most column then the cursor will wrap around to the first character position on the next line. The entire layer is scrolled up one line if the cursor was at the bottom of the layer. When a DMD function key is pressed, the code for that function (stored in battery backed RAM) is sent to the host.

Mouse Buttons

Buttons one and three function as they normally do in a layer that has had nothing downloaded into it. Button two is used to display the *hp2621* menu with the following options:

backup The *hp2621* emulator has 4096 bytes of on- and off-screen storage. As lines scroll off the top edge of the layer, they are maintained in this memory. The **backup** option allows the user to page backward through this memory. Each time this option is selected, the information displayed in the layer is replaced by information one page backward. A page is defined for scrolling purposes as roughly 3/4 of the length of the layer.

forward The forward menu option is used to page forward through display memory. Both **backup** and **forward** will leave at least 2 blank lines at the bottom of the layer.

reset This option resets memory such that the last page of data is the page that is being displayed in the layer. Again, a page is about 3/4 the length of the layer.

rev vid Selecting this menu option toggles reverse video within the layer.

blink This option toggles the cursor between a solid rectangle to a blinking rectangle.

clear **clear** produces a blank layer. It does not alter data stored in display memory. Selecting the menu options **backup, forward,** or **reset** will redisplay data in memory.

24x80 This option will reshape the current layer such that it is 24 rows long by 80 columns wide. This option must always be used when working with the *vi* editor. If there is not enough memory to do the reshaping then the layer is reshaped, to a small rectangle in the upper left-hand corner of the screen.

page/scroll Normally, the layer is scrolled - lines are moved up one row at a time as line feeds are encountered on the bottom line of the layer. Selecting **page** will cause input to halt at the bottom of the layer until a character is entered from the keyboard. The layer is then cleared and output continues at the top of the layer.

The menu option is changed from **page** to **scroll.** Reselecting scroll switches the emulator back to scrolling mode and changes the menu option back to **page.**

new **new** allows the user to create a new layer (without the need to download the emulator a second time) that will also run the emulator. Button three is used as normal for sweeping out the new layer. If specified in the command line, the *startup* command is sent to the host within this new layer.

FILES

$DMD/bin/hp2621	Shell script for downloading emulator.
$DMD/lib/hp2621.m	Emulator program.

SEE ALSO

layers(7).
stty(1), vi(1) in the *UNIX System V User Reference Manual.*
term(7) in the *UNIX System V Administrator Reference Manual.*

BUGS

The emulator does not examine *stty*(1) options. Therefore, such options as arbitrary tab settings are not available.

NAME

icon – interactive icon and picture drawer

SYNOPSIS

icon [**–x** m] [**–y** n]

DESCRIPTION

The *icon* utility is a menu-driven interactive icon and picture drawing program. It runs under *layers* using the "mouse" and keyboard for command and text entry. The default *icon* display consists of a 50x50 cell grid in the lower right-hand corner of the layer in which *icon* was invoked. By invoking *icon* with the **–x** and **–y** flags, the grid size may be specified to be *m*X*n*, overriding the default. Each cell in this grid corresponds to a single bit in the *icon* being created or edited. In the upper left hand corner of the layer, an actual size view of the icon is displayed.

Button 1 Button 1, when depressed and held in, draws a black square in the grid position pointed to by the mouse cursor.

Button 2 Button 2, when depressed and held in, undraws any black square in the grid position pointed to by the mouse cursor.

Button 3 Button 3, when depressed, displays a menu of icons. By moving the cursor (now a box) over the desired icon and releasing the button, a command will be invoked. If the button 3 command requires a section of the grid display to be selected, hitting button 2 will format a 16x16 grid outline. To specify other than this 16x16 grid outline, hit button 3 and sweep out the rectangle you wish the command to act on.

COMMANDS

The icons menu are described below from the upper left by rows to the bottom right. On the bottom row is a help command designated by the word "help".

Arrow move to another portion of the grid.

Copier copy to another portion of the grid.

Invert	change lights to darks and darks to lights (invert video).
Trash Can	erase.
Reflect X	flip on the x-axis.
Reflect Y	flip on the y-axis.
Rotate +	rotate 90 degrees clockwise.
Rotate -	rotate 90 degrees counterclockwise.
Shear X	shear along the x-axis.
Shear Y	shear along the y-axis.
Stretch	stretch (expand).
Texture 16	take one pattern and make many copies of it.
Glasses	read an icon file.
Grid	draw a reference grid.
Mouse	change current mouse cursor to icon.
Quill pen	write an icon file.
Grid, arrow to grid	bitblt operator.
HELP	prints the help menu.
Smoking Gun	exit the *icon* program.

CURSOR ICONS

The following are status indicator icons that the mouse cursor changes to under various conditions:

Alarm clock	wait.
Dead Mouse	mouse inactive.
Dark square in stack	menu on button 3.
Square with arrow	sweep rectangle (button 3).
Double square with arrow	sweep rectangle (button 3) or get 16x16 grid frame (button 2).

FILES

$DMD/lib/icon.m	terminal support program

```
$DMD/icon/*              icons
$DMD/icon/texture/*      textures
$DMD/icon/texture16/*    texture16s
```

SEE ALSO

layers(1).

BUGS

Due to the amount of available memory in the DMD, the number of programs currently executing in the DMD, and the size of the *icon* grid, the terminal may lock after *icon is downloaded.*

NAME

ismpx – query terminal state

SYNOPSIS

ismpx [-]

DESCRIPTION

The *ismpx* utility is used to determine whether your DMD 5620 is under *layers* mode or *stand-alone* mode. It is useful for shell scripts that download DMD programs, such as *demo*(1).

The *ismpx* utility prints "yes" and returns 0 (true) if invoked under *layers*, and prints "no" and returns 1 otherwise. When invoked with the dash "–", *ismpx* does not print anything and just returns with the proper exit code.

SEE ALSO

demo(1), layers(1).

EXAMPLE

```
if ismpx -
then
        32ld bounce.m
else
        32ld bounce.j
fi
```

NAME

jim, jim.recover – DMD text editor

SYNOPSIS

jim [files]
jim.recover [**–m**] [**–f**] [**–t**] [files]

DESCRIPTION

jim is the text editor for the DMD terminal. It relies on the mouse to select text and commands. It runs only under *layers*(1). *jim*'s screen consists of a number of *frames*, a one-line command and diagnostic frame at the bottom, and zero or more larger file frames above it. Except where indicated, these frames behave identically. One of the frames is always the current frame, to which typing and editing commands refer, and one of the file frames is the working frame, to which file commands such as pattern searching and I/O refer.

A frame has at any time a selected region of text, which is indicated by reverse video highlighting. The selected region may be a null string between two characters, indicated by a narrow vertical bar between the characters. The editor has a single *save buffer* containing an arbitrary string. The editing commands simply invoke transformations between the selected region and the save buffer.

The mouse buttons are used for the most common operations. Button 1 (left) is used for selection. Clicking button 1 in a frame which is not the current frame makes the indicated frame current. Clicking button 1 in the current frame selects the null string closest to the mouse cursor. Making the same null selection twice ('double clicking') selects (in decreasing precedence) the bracketed or quoted string, word or line enclosing the selection. By pushing and holding button 1, an arbitrary contiguous visible string may be selected. Button 2 provides a small menu of text manipulation functions that are described below. Button 3 provides control for inter-frame operations.

The button 2 menu entries are:

cut Copy the selected text to the save buffer and delete it from the frame. If the selected text is null, the save buffer is unaffected.

paste Replace the selected text by the contents of the save buffer.

snarf Copy the selected text to the save buffer. If the selected text is null, the save buffer is unaffected.

Typing replaces the selected text with the typed text. If the selected text is not null, the first character typed forces an implicit **cut**. Control characters are discarded, but BS (control H), ETB (control W), LF (linefeed) and ESC (escape) have special meanings. BS is the usual backspace character, which erases the character before the selected text (which is a null string when it takes effect). ETB erases back to the word boundary preceding the selected text. There is no line kill character. LF toggles the current frame between the workframe and the diagnostic frame, and can be a substitute for manual frame selection with the mouse. ESC selects the text typed since the last button hit or ESC. If an ESC is typed immediately after a button hit or ESC, it is identical to a **cut**. ESC followed by **paste** provides the functionality of a simple undo feature.

The button 3 menu entries are:

new Create a new frame, much as in *layers*.

reshape
Change the shape of the indicated frame, much as in *layers*. The frame is indicated by a button 3 hit after the selection.

close Close the indicated frame and its associated file. The file is still available for editing later; only the associated frame is shut down.

write Write the indicated frame's contents to its associated file.

The rest of the menu is a list of file names available for editing. To work in a different file, select the file from the menu. If the file is not open on the screen, the cursor will switch to an outline box to prompt for a rectangle to be swept out with button 3, as in the New operator of *layers*. (Unlike *layers*, there is a shorthand: sweeping the empty rectangle creates the largest possible rectangle.) If the file is already open, it will simply be made the workframe and current frame (for typing), perhaps after redrawing if it

is obscured by another frame. The format of the lines in the menu is:

- possibly an apostrophe, indicating that the file has been modified since last written,
- possibly a period or asterisk, indicating the file is open (asterisk) or the workframe (period),
- a blank,
- and the file name. The file name may be abbreviated by compacting path components to keep the menu manageable, but the last component will always be complete.

The work frame has a *scroll bar* — a black vertical bar down the left edge. A small tick in the bar indicates the relative position of the frame within the file. Pointing to the scroll bar and clicking a button controls scrolling operations in the file:

button 1
: Move the line at the top of the screen to the y position of the mouse.

button 2
: Move to the absolute position in the file indicated by the y position of the mouse.

button 3
: Move the line at the y position of the mouse to the top of the screen.

The bottom line frame is used for a few typed commands, modeled on *ed*(1), which operate on the workframe. When a carriage return is typed in the bottom line, the line is interpreted as a command. The bottom line scrolls, but only when the first character of the next line is typed. Thus, typically, after some message appears in the bottom line, a command need only be typed; the contents of the line will be automatically cleared when the first character of the command is typed. The commands available are:

e *file*
: Edit the named *file*, or use the current file name if none specified. Note that each file frame has an associated file name.

E *file* Edit the named *file* unconditionally, as in *ed*(1).

f *file* Set the name of the file associated with the work frame, if one is specified, and display the result.

g *files* Enter the named *files* into the filename menu, without duplication, and set the work frame to one of the named files. If the new work frame's file is not open, the user is prompted to create its frame. The arguments to are passed through *echo*(1) for shell metacharacter interpretation.

w *file* Write the named *file*, or use the current file name if none specified.

q Quit the editor.

Q Quit the editor unconditionally, as in *ed*(1).

/ Search forward for a string matching the regular expression after the slash. If found, the matching text is selected. The regular expressions are exactly as in *egrep*(1), with two additions: the character '@' matches any character *including* newline, and the sequence '\n' specifies a newline, even in character classes. The negation of a character class does not match a newline. An empty regular expression (slash-newline) repeats the last regular expression.

? Search backwards for the expression after the query.

94 Select the text of line 94, as in *ed*.

$ Select the text of the last line.

cd *dir* Set the working directory to *dir*, as in the shell. There is no CDPATH search, but $HOME is the default *dir*.

= Display the line number of selection in the current frame.

>*UNIX-command*
Send the selected text to the standard input of *UNIX-command*.

<*UNIX-command*
Replace the selected text by the standard output of *UNIX-command*.

| *UNIX-command*

Replace the selected text by the standard output of *UNIX-command*, given the original selected text as standard input.

If any of <, > or | is preceded by an asterisk *, the command is applied to the entire file, instead of just the selected text. If the command for < or | exits with non-zero status, the original text is not deleted; otherwise, the new text is selected. Finally, the standard error output of the command, which is merged with the standard output for >, is saved in the file $HOME/jim.err . If the file is non-empty when the command completes, the first line is displayed in the diagnostic frame. Therefore the command ">pwd" will report *jim*'s current directory.

The most recent search command ('/' or '?') and UNIX command ('<', '|', or '>') are added to the button 2 menu, so that they may be easily repeated.

Attempts to quit with modified files, or edit a new file in a modified frame, are rejected. A second 'q' or 'e' command will succeed. The 'Q' or 'E' commands ignore modifications and work immediately. Some consistency checks are performed for the 'w' command. *jim* will reject write requests which it considers dangerous (such as writes which would change a file modified since read it into its memory). A second 'w' will always write the file.

If *jim* receives a hangup signal, it writes a file *jim.recover*, which is a shell command file that, when executed, will retrieve the files that were being edited. If it cannot write this file in the current directory, it writes it in /tmp. The **–m** option recovers only modified files; **–t** prints a table of contents. By default, *jim.recover* is interactive; the **–f** option suppresses the interaction. If no files are named to *jim.recover*, it will recover all files open at the time *jim* was killed.

jim is reshapeable, but a reshape frees the screen space for all open frames.

FILES

$DMD/lib/jim.m	terminal support program
/tmp/jim.*	temporary file
$HOME/jim.err	diagnostic output from UNIX commands

jim.recover recovery script created upon *jim* failure

BUGS

The regular expression matcher is non-deterministic (unlike *egrep*), and may be slow for complicated expressions. The < and ¦ operators do not snarf the original text.

NAME

jim.recover – interactively recover lost editing after abnormal failure

SYNOPSIS

jim.recover [**–m**] [**–f**] [files ...]

DESCRIPTION

The *jim.recover* utility is created in the current directory by *jim*(1) whenever an abnormal failure occurs (e.g., *jim*'s layer gets deleted). Its purpose is to allow the user to determine the disposition of edits made before the *jim* failure. One by one, the names of each file being edited at the time of failure will be printed along with an indication of whether the file was modified since last written. Responding 'y' or 'Y' will cause the edits for that file to be saved; otherwise, the changes will be lost.

The options are:

–m looks only at files modified since the last write (i.e., only modified files).

–f forces all files to appear modified.

If any files are specified, only the named files will be considered; the default is for all files to be considered.

FILES

$DMD/bin/jimunpack tool called by *jim.recover*

SEE ALSO

jim(1), layers(1).

NAME

jterm – reset DMD layer

SYNOPSIS

jterm

DESCRIPTION

The *jterm* utility is used to reset the DMD 5620 terminal after invoking *tplot, proof,* or other programs that change the terminal attributes. It is useful only under *layers*(1). In practice, it is most commonly used to restart the default terminal emulator after using an alternate, such as *hp2621* or *tek4014*. In *stand-alone* mode, rebooting the terminal should be used to restore the normal terminal program.

SEE ALSO

hp2621(1), layers(1), proof(1), tek4014(1), tplot(1G).

NAME

jwin – print size of layer

SYNOPSIS

jwin

DESCRIPTION

Jwin runs only under *layers*(1) and is used to determine the size of the current layer. *Jwin* prints 2 lines. The first line shows the width and the height of the layer in bytes (number of characters across and number of lines, respectively). The second line shows the width and height of the layer in bits.

SEE ALSO

layers(1).

EXAMPLE

```
$ jwin
bytes:  86 25
bits:   780 406
$
```

NAME

jx – DMD execution and stdio interpreter

SYNOPSIS

jx file
... ! **jx** file

DESCRIPTION

The *jx* utility downloads the program in *file* to the DMD on **/dev/tty** and runs it there, simulating most of the standard I/O library functions. *Stdout* and *stderr* are properly redirected, while *stdin* is properly redirected only if it is not from the keyboard. Programs wishing to read from the keyboard should use *kbdchar*().

Stdout and *stderr*, if directed to a controlling DMD, will be stored in a buffer during execution. After the terminal program has been rebooted, *stdout* and *stdin* will be redirected to the terminal.

Programs intended for use by *jx* should include <**dmd.h**> and call *exit*() upon termination. *Exit*() returns control to the shell and causes a reboot of the ROM terminal program. *Jx* programs should be compiled with the **–J** option of *dmdcc* for stand-alone operation.

Stdio functions available under *jx* are:

getc	getchar	fgets	fflush
putc	putchar	puts	fputs
fopen	freopen	fclose	access
popen	pclose	fread	fwrite
printf	sprintf	fprintf	

Printf is stripped down. It has the %d, %s, %c, %o, %x, and %u conversions (each of which performs the same conversion as the C language printf conversion by the same name), and the %D conversion, which prints an unsigned long decimal number.

FILES

$DMD/include/dmdio.h	
$DMD/lib/sysint	standard I/O interpreter
$HOME/.jxout	saved standard output
$HOME/.jxerr	saved standard diagnostic output

SEE ALSO

dmdcc(1).
access(2), fopen(3S), fread(3S), getc(3S), popen(3S), printf(3S), putc(3S), puts(3S) in the *UNIX System V Programmer Reference Manual*.

BUGS

Keyboard standard input does not work. *Jx* does not work when su'ed to another user.

NAME

layers – layer multiplexor for DMD

SYNOPSIS

layers [**–s**] [**–t**] [**–f** file] [terminal program]

DESCRIPTION

The *layers* utility manages asynchronous windows (layers) on the DMD terminal. Upon invocation, it loads the DMD with a terminal program that is the primary user interface. DMD should be set in the user's *.profile* and exported. Then it allocates a new channel group to access the *xt* driver, connects the channels to the standard output, and listens on channel 0 for commands from the DMD.

There are three flags:

–s Produces statistics at the end of the session. The statistics may be printed during a session by invoking the program *xts*(1).

–t Turn on driver packet tracing, and produce a trace dump at the end of the session. The trace dump may be printed during a session by invoking the program *xtt*(1).

–f *file* Create new layers as specified in the named *file*. Each line of the file represents one new layer. It contains the rectangle coordinates ordered, origin.x, origin.y, corner.x, corner.y, followed by the command to be executed in the layer. The separators are space or tab. The last line of the file will be the current layer when it is brought up. The screen is {8, 8, 792, 1016}. This enforces the same 8 pixel border that making layers with the mouse does. Layers which have an origin outside the border will be moved inside.

Each layer is in most ways functionally identical to a separate terminal. Characters typed on the keyboard are sent to the standard input of the UNIX system process running in the layer, and characters written on the standard output appear in the layer. When a layer is created, a separate shell is established, and bound to the layer. If the environment variable **SHELL** is set, the user will get

that shell; otherwise, the *sh* shell will be used. In order to enable communications with other users via *write*(1), *layers* invokes the command *relogin*(1) when the first layer is created. *Relogin* will reassign the layer as the user's logged-in terminal. An alternative layer can be designated by using *relogin* directly. The *layers* utility will restore the real line on termination. The maximum number of *layers* allowed is 6.

Layers are created, deleted, and otherwise dealt with using the mouse on the DMD. Menu items consist of **New**, **Reshape**, **Top**, **Bottom**, **Current**, **Delete**, **Move**, and **Exit**. Depressing button 3 activates a menu of *layer* operations. Lifting button 3 then selects an operation. At this point, the *target* cursor indicates that an operation is pending. Hitting button 3 again activates the operation on the layer pointed to by the cursor.

New The only menu item to give a *sweep* cursor upon selection, and is used to create a layer. It requires a rectangle to be swept out across any diagonal while button 3 is depressed. The *sweep* cursor indicates that a rectangle is to be created.

Reshape Changes the size and location of a layer on the screen. Selecting **Reshape** results in a *target* cursor. Move the cursor to the window to be reshaped, click button 3 again, and the cursor becomes the *sweep* cursor. Now the window can be reshaped by redrawing the window just as in the **N**ew command.

Move Relocates a layer without changing its shape or data. Selecting **Move** results in a *target* cursor. Clicking button 3 on the layer to be relocated changes the cursor to *multi directional* arrows. The layer may now be relocated by using the mouse.

Top Makes the layer pointed at by the *target* cursor the top layer on the screen.

Bottom Places the layer pointed at by the *target* cursor as the bottom layer of the screen.

Current Makes the layer pointed at by the *target* cursor available for keyboard input.

Delete Deletes a window which is not the current window.

Exit The *skull and crossbones* cursor indicates that executing this operation by clicking button 3 again will exit from the *layers* program.

Button 1 is a shorthand for **C**urrent and **T**op, which pulls a layer to the front of the screen and makes it the current layer for keyboard input. Layers which are not current are indicated by having a smaller border.

SEE ALSO

32ld(1), jx(1), relogin(1), xts(1), xtt(1), xt(4).
sh(1), write(1) in the *UNIX System V User Reference Manual.*

WARNINGS

Reshape only works properly for processes that check to see if they have been reshaped.

When invoking *layers* with the **–s** and/or the **–t** options, it is best to redirect *stderr* to another file to save the statistics and tracing output (e.g., **layers –s 2>stats**), otherwise, all or some of the output may be lost.

NAME

lens – DMD interactive magnified viewing window

SYNOPSIS

lens

DESCRIPTION

The *lens* program allows the user to magnify any portion of the DMD screen. Magnification is done via a rectangle which takes the place of the mouse cursor. The rectangle acts as a window and anything appearing in the window is displayed magnified in a companion window. The user has the option to select the size and shape of the viewing window and the amount of magnification. Magnification can be anywhere from 1x up to 50x.

MOUSE

The mouse is used to move a viewing window around the screen and to select menus options. Button 3 is the same as button 3 used by layers. Button 1 is used as a toggle to turn the magnifying window on and off. Button 2 is used to select menu options pertaining to *lens*. The following is a description of these menu choices:

go, stop This menu selection is used for starting (*go*) and stopping (*stop*) the magnification process. When first selected, two rectangles are displayed where the menu was. The smaller rectangle (50x50 pixels) is used to select the area to be viewed. As the mouse is moved, the rectangle moves and thereby selects a different section of the screen to magnify. The larger rectangle is the magnified view of the smaller rectangle. The larger rectangle is initially twice the size (2x) of the viewing window.

The next selection of this menu option will turn off the magnification process. The mouse cursor will be restored to the icon used by the current layer in which the cursor is located.

Button 1 is a short cut method for turning the *lens* off and on. The first depression of the button reverses the previous state of the *lens*. For example, if the *lens* was turned on by the button 2 menu, then

pressing button 1 will turn the *lens* off.

bigger *Bigger* is used to increase the size of the magnify window, thereby increasing the amount of the magnification. Each time the option is selected, the window is increased by the sum of the last two increases, that is, the order of magnifications is 1x, 2x, 3x, 5x, 8x, and 13x. The size of the magnification is limited to 13x.

smaller *Smaller* reverses the magnifications performed by *bigger*. The magnification rectangle is decreased in size in the reverse order of that performed by *bigger*. The window is reduced by one increment each time the *smaller* menu option is selected. The smallest magnification is 1x. Subsequent selections of *smaller* after 1x have no affect.

new viewer This option allows the user to define his or her own *lens* size. After selecting this option, a box with an arrow in it is displayed where the menu used to be. Button 3 is then used to sweep out the new viewing rectangle. If either button 1 or 2 is pressed, then the process of defining a new window is canceled. Button 1, however, will still act as a toggle for turning the *lens* on and off. Selecting a new viewer will always delete the old one.

pause This option is only available when the viewing and magnification windows are active. It temporarily (until a button is pressed) stops the two windows from tracking the mouse. This can be useful when you want to view something without the worry of the mouse moving. Pressing any button will turn *pause* off.

exit This option is used to exit the program and to display the setting (rising?) sun, giving the user a chance to cancel the request. Buttons 1 and 2 will cancel the request; button 3 will execute it.

BUGS

Magnifying a very small window as large as possible does not work properly. A magnifying window does get created but no magnified image gets displayed in it.

FILES

$DMD/lib/lens.m	terminal support program
$DMD/bin/lens	shell script for executing lens

NAME

lpg – interface for bitmap printing on the lp spooler system

SYNOPSIS

lpg file

DESCRIPTION

The *lpg* utility is a shell script that sends the output to the system printer. It was written to get around the problem of getting graphics output via the spooler. Whenever processing is complete, a file named **file.g** will be created. This file should be removed when the hardcopy is obtained.

SEE ALSO

screendump(1), bcan(1).

NAME

m32as – *WE*-32001 processor module assembler

SYNOPSIS

m32as [**–o** objfile] [**–n**] [**–m**] [**–V**] file-name

DESCRIPTION

The *m32as* command assembles the named file.

The following flags are recognized by the assembler and may be specified in any order:

–o *objfile* Output of assembly is put in *objfile*. By default, the output file name is formed by removing the **.s** suffix, if there is one, from the input file name and appending a **.o** suffix.

–n Turns off long/short address optimization. By default, address optimization takes place.

–m Invoke the *m4* macro processor. By default, do not invoke *m4* on the input to the assembler.

–V Causes the version number of the assembler being run to be written on standard error.

FILES

/usr/tmp/m32as[1-6]*XXXXXX* temporary files

SEE ALSO

dmdcc(1), m32ld(1), m32nm(1), m32strip(1).
m4(1) in the *UNIX System V User Reference Manual.*

DIAGNOSTICS

If the input file cannot be read, the assembly will terminate with the message, " Unable to open input file" . If assembly errors are detected in the input file, the following information is written to standard error: the input file name, line number where the error occurred in the assembly code, a (hopefully) descriptive message of the problem, and, if the input file was produced by the C compiler [see *m32cc*(1)], the line number in the C program that generated the erroneous code.

WARNINGS

If the input file does not contain a **.file** assembler directive, then the file name given by the assembler, when an error occurs, is one

of the temporary files.

BUGS

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

NAME

m32conv – *WE*-32001 processor module Software Generation System (SGS) object file converter

SYNOPSIS

m32conv [-] [-p] [-s] -t target files

DESCRIPTION

The *m32conv* command converts *WE*-32001 processor module object files from their current format to the format of the *target* machine. The converted file is written to file.*v*.

Command line options are:

– indicates *files* should be read from *stdin*.

–p makes convert produce the output file in the portable archive format (if the input file is an archive in either the old or new format). Otherwise, all archives will be produced in the old format.

–s causes *m32conv* to function exactly as 3bswab, i.e., to " preswab" all characters in the object file. This is useful only for AT&T 3B20 computer object files which are to be " swab-dumped" from a DEC machine to a 3B20 computer.

–t target indicates the machine (target) to which the object file is being shipped. This may be another host or a target machine. Legal values for **target** are: pdp, vax, ibm, i80, x86, b16, n3b and m32.

The *m32conv* utility can be used to convert all object files in common object file format, not only *WE*-32001 processor module object files. It can be used on either the source (" sending") or target (" receiving") machine.

The *m32conv* utility is meant to ease the problems created by a multi-host, cross-compilation development environment. *m32conv* is best used within a procedure for shipping object files from one machine to another.

DIAGNOSTICS

The diagnostics are all intended to be self-explanatory. Fatal diagnostics on the command lines cause termination. Fatal diagnostics

on an input file cause the program to continue to the next input file.

BUGS

Special applications must compile *m32conv* differently if it is to convert special object files, e.g., products of *ldp*, correctly. Archives created on another architecture and converted on the target machine may have members with incorrect permissions (this has no effect so long as they remain members of the archive).

NAME

m32cprs – compress an IS25 object file

SYNOPSIS

m32cprs [**–pv**] infile outfile

DESCRIPTION

The *m32cprs* command reduces the size of an IS25 object file, *infile*, by removing duplicate structure and union descriptors. The reduced file, *outfile*, is produced as output.

The options are:

–p Print statistical messages including:

total number of tags
total duplicate tags
total reduction of *infile*.

–v Print verbose error messages if error condition occurs.

EXAMPLE

```
m32cprs dmda.out sm3b
```

SEE ALSO

m32strip(1).

NAME

m32dis – *WE*-32001 processor module disassembler

SYNOPSIS

m32dis [**–o**] [**–V**] [**–L**] [**–d** sec] [**–da** sec] [**–F** function] [**–t** sec] [**–l** string] files

DESCRIPTION

The *m32dis* command produces an assembly language listing of each of its object *file* arguments. The listing includes assembly statements and the binary that produced those statements.

The following *options* are interpreted by the disassembler and may be specified in any order.

–o Will print numbers in octal. Default is hexadecimal.

–V Version number of the disassembler will be written to standard error.

–L Invokes a lookup of **C** source labels in the symbol table for subsequent printing.

–d sec Disassembles the named section as data, printing the offset of the data from the beginning of the section.

–da sec Disassembles the named section as data, printing the actual address of the data.

–F function Disassembles single named functions in each object file that is specified on the command line.

–t sec Disassembles the named section as text.

–l string Will disassemble the library file specified as *string*. For example, one would issue the command:

```
m32dis -l x -l z
```

to disassemble **libx.a** and **libz.a**. On the 3B20 and VAX computers, all libraries are assumed to be in **$DMD/m32/lib**. On the 3B2 and 3B5 computers, the libraries are assumed to be in **/lib**.

If the **–d**, **–da** or **–t** options are specified, only those named sections from each user-supplied file name will be disassembled.

Otherwise, all sections containing text will be disassembled.

If the **–F** option is specified, only those named functions from each user-supplied file name will be disassembled.

On output, a number enclosed in brackets at the beginning of a line, such as [**5**], represents that the **C** breakpointable line number starts with the following instruction. An expression such as <**40**> in the operand field, following a relative displacement for control transfer instructions, is the computed address within the section to which control will be transferred. A **C** function name will appear in the first column, followed by **( )**.

SEE ALSO

m32as(1), m32cc(1), m32ld(1).

DIAGNOSTICS

The self explanatory diagnostics indicate errors in the command line or problems encountered with the specified files.

NAME

m32dump – dump selected parts of an object file

SYNOPSIS

m32dump [**–a**] [**–f**] [**–o**] [**–h**] [**–s**] [**–r**] [**–l**] [**–t**] [**–z** name] files

DESCRIPTION

The *m32dump* command dumps selected parts of each of its object *file* arguments.

This command will accept both object files and archives of object files. It processes each file argument according to one or more of the following *options:*

–a Dump the archive header of each member of each archive file argument.

–f Dump each file header.

–o Dump each optional header.

–h Dump section headers.

–s Dump section contents.

–r Dump relocation information.

–l Dump line number information.

–t Dump symbol table entries.

–z name Dump line number entries for the named function.

The following *modifiers* are used in conjunction with the *options* listed above to modify their capabilities.

–d number Dump the section number or range of sections starting at *number* and ending either at the last section number or *number* specified by +**d**.

+**d** number Dump sections in the range either beginning with first section or beginning with section specified by **–d**.

–n name Dump information pertaining only to the named entity. This *modifier* applies to **–h**, **–s**, **–r**, **–l**, and **–t**.

–t index Dump only the indexed symbol table entry. The **–t** used in conjuction with +**t**, specifies a range of symbol table entries.

+**t** index Dump the symbol table entries in the range ending with the indexed entry. The range begins at the first symbol table entry or at the entry specified by the **–t** option.

–v Dump information in symbolic representation rather than numeric (e.g., C_STATIC instead of **0X02**). This *modifier* can be used with all the above *options* except **–s** and **–o** options of *m32dump*.

–z name,number
Dump line number entry or range of line numbers starting at *number* for the named function.

+**z** number Dump line numbers starting at either function *name* or *number* specified by **–z,** up to *number* specified by +**z**.

Blanks separating an *option* and its *modifier* are optional. The comma separating the name from the number modifying the **–z** option may be replaced by a blank.

The *m32dump* command attempts to format the information it dumps in a meaningful way, printing certain information in character, hex, octal or decimal representation as appropriate.

NAME

m32ld – link editor for *WE*-32001 processor module object files

SYNOPSIS

m32ld [**–a**] [**–e** epsym] [**–f** fill] [**–l**x] [**–m**] [**–r**] [**–s**] [**–o** outfile] [**–u** symname] [**–L** dir] [**–N**] [**–V**] [**–VS** num] [**–X**] file-names

DESCRIPTION

The *m32ld* command combines several object files into one, performs relocation, resolves external symbols, and supports symbol table information for symbolic debugging. In the simplest case, the names of several object programs are given, and *m32ld* combines them, producing an object module that can either be executed or used as input for a subsequent *m32ld* run. The output of *m32ld* is left in **m32a.out**. This file is executable if no errors occurred during the load. If any input file, *file-name*, is not an object file, *m32ld* assumes it is either an ASCII file containing link editor directives or an archive library.

If any argument is a library, it is searched exactly once at the point it is encountered in the argument list. Only those routines defining an unresolved external reference are loaded. If a routine from a library references another routine in the library, the referenced routine must appear after the referencing routine in the library. Thus the order of programs within libraries is important.

The following options are recognized by *m32ld*.

–a Produces an absolute file and gives warnings for undefined references. Relocation information is stripped from the output object file unless the **–r** option is given. The **–r** option is needed only when an absolute file should retain its relocation information (not the normal case). If neither **–a** nor **–r** is given, **–a** is assumed.

–e epsym Sets the default entry point address for the output file to be that of the symbol *epsym*. This option forces the **–X** option to be set.

–f fill This option sets the default fill pattern for "holes" within an output section as well as initialized bss sections. The argument *fill* is a two-byte constant.

–l This option specifies a library named *x*. It stands for **lib*x*.a** where *x* is up to seven characters. A library is searched when its name is encountered, so the placement of a **–l** is significant. By default, libraries are located in LIBDIR.

–m This option causes a map or listing of the input/output sections to be produced on the standard output.

–o outfile This option produces an output object file by the name *outfile* . The name of the default object file is **dmda.out**.

–r This option causes relocation entries to be retained in the output object file. Relocation entries must be saved if the output file is to become an input file in a subsequent *m32ld* run. Unless **–a** is also given, the link editor will not complain about unresolved references.

–s This option causes line number entries and symbol table information to be stripped from the output object file.

–u symname Takes the argument *symname* as a symbol and enters it as undefined in the symbol table. This is useful for loading entirely from a library, since initially the symbol table is empty and an unresolved reference is needed to force the loading of the first routine.

–L dir Changes the algorithm of searching for **lib*x*.a** to look in *dir* before looking in LIBDIR.

–N Puts the data section immediately following the text in the output file

–V Outputs a message giving information about the version of m32ld being used.

–VS num The **num** argument is taken as a decimal version number identifying the **m32a.out** file that is produced. The version stamp is stored in the optional header.

–X Generates a standard UNIX system file header within the "optional header" field in the output file.

FILES

LIBDIR/lib*x*.a	libraries
dmda.out	output file

SEE ALSO

dmda.out(1).

WARNINGS

Through its input directives, the link editor gives users great flexibility; however, people who use the input directives must assume some added responsibilities. Input directives should ensure the following properties for programs:

– C defines a zero pointer as null. A pointer to which zero has been assigned must not point to any object. To satisfy this, users must not place any object at virtual address zero in the data space.

NAME

m32list – produces C source listing from *WE*-32001 processor module object file

SYNOPSIS

m32list [**–V**] [**–h**] source-file . . . [object-file]

DESCRIPTION

The *m32list* command produces a C source listing with line number information attached. If multiple C source files were used to create the object file, *m32list* will accept multiple file names. The object file is taken to be the last non-C source file argument. If no object file is specified, the default object file, **m32a.out**, will be used.

Line numbers will be printed for each breakpoint inserted by the compiler (generally, each executable C statement that begins a new line of source). Line numbering begins anew for each function. Line number 1 is always the line containing the left curly brace ({) that begins the function body. Line numbers will also be supplied for inner block redeclarations of local variables so that they can be distinguished by the symbolic debugger.

The **–V** flag will supply version information of the *m32list* command.

The **–h** flag will suppress heading output.

WARNINGS

Object files given to *m32list* must have symbolic debugging symbols.

Since *m32list* does not use the C preprocessor, it may be unable to recognize function definitions whose syntax has been distorted by the use of C preprocessor macro substitutions.

SEE ALSO

m32as(1), m32cc(1), m32ld(1)

DIAGNOSTICS

"m32list: name: cannot open" if *name* cannot be read.

NAME

m32lorder – find ordering relation for an object library

SYNOPSIS

m32lorder files

DESCRIPTION

The input is one or more object or library archive *files* [see *ar*(1)]. The standard output is a list of pairs of object file names meaning that the first file of the pair refers to external identifiers defined in the second. The output may be processed by *tsort*(1) to find an ordering of a library suitable for one-pass access by *m32ld(1)*.

The following example builds a new library from existing **.o** files.

```
ar cr library *'m32lorder *.o | tsort*'
```

FILES

*symref, *symdef temporary files

SEE ALSO

m32ld(1).
ar(1), tsort(1) in the *UNIX System V User Reference Manual.*

BUGS

Object files whose names do not end with **.o**, even when contained in library archives, are overlooked. Their global symbols and references are attributed to some other file.

NAME

m32nm –print name list of *WE*-32001 object file

SYNOPSIS

m32nm [**–o** ¦ **–h** ¦ **–d**] [**–v**] [**–n**] [**–e**] [**–a**] [**–f**] [**–u**] [**–V**] file-name . . .

DESCRIPTION

The *m32nm* command displays the symbol table of each *WE*-32001 object file *file-name*. *File-name* may be a relocatable or absolute *WE*-32001 object file or it may be an archive of such *WE*-32001 object files. For each symbol, the following information will be printed:

Name The name of the symbol.

Value Its value expressed as offset or an address depending on its storage class.

Class Its storage class.

Type Its type and derived type. If the symbol is an instance of a structure or of a union, then the structure or union tag will be given following the type (e.g., struct-tag). If the symbol is an array, then the array dimensions will be given following the type (e.g., **char**[n][**m**]

Size Its size in bytes, if available.

Line The source line number at which it is defined, if available.

Section For storage classes static and external, the object file section containing the symbol.

The output of *m32nm* may be controlled using the following flags:

–h A symbol's value and size will be printed in hexadecimal (default).

–d A symbol's value and size will be printed in decimal instead of hexadecimal.

–o A symbol's value and size will be printed in octal instead of hexadecimal.

–v External symbols will be sorted by value before they are printed.

–n External symbols will be sorted by name before they are printed.

–e Only static and external symbols are printed.

–a Full output is produced. Redundant symbols (.text, .data, and .bss), normally suppressed, are printed.

–f "Fancy" output is produced; that is, the symbol table information is post-processed to reflect the block structure of the source code.

–u Only undefined symbols are printed.

–V Version of *m32nm* command executing.

Flags may be used in any order, either singly or in combination, and may appear anywhere in the command line. Therefore, both

```
m32nm name -e -v
```

and

```
m32nm -ve name
```

print the static and external symbols in *name*, with external symbols sorted by value.

FILES

/usr/tmp/nm??????

SEE ALSO

m32as(1), m32cc(1), m32ld(1).

DIAGNOSTICS

"m32nm: name: cannot open"
if *name* cannot be read.

"m32nm: name: bad magic"
if *name* is not a *WE*-32001 object file.

"m32nm: name: no symbols"
if the symbols have been stripped from *name*.

NAME

m32size – print section sizes of *WE*-32001 object files

SYNOPSIS

m32size [**-o**] [**-d**] [**-V**] files

DESCRIPTION

The *m32size* command produces section size information for each section in the *WE*-32001 object files. For each section in the object file, the name of the section is printed followed by its size in bytes, its physical address, and its virtual address.

Numbers will be printed in hexadecimal unless either the **–o** or the **–d** option is used, in which case they will be printed in octal or in decimal, respectively.

The **–V** flag will supply the version information on the *m32size* command.

SEE ALSO

m32as(1), m32cc(1), m32ld(1).

DIAGNOSTICS

" m32size: name: cannot open"
if *name* cannot be read.

" m32size: name: bad magic"
if *name* is not a *WE*-32001 object file.

NAME

m32strip – strip symbol and line number information from *WE*-32001 processor module object file

SYNOPSIS

m32strip [**–l**] [**–x**] [**–r**] [**-V**] file-names

DESCRIPTION

The *m32strip* command strips the symbol table and line number information from *WE*-32001 processor module object files, including archives. Once this has been done, no symbolic debugging access will be available for that file; therefore, this command is normally run only on production modules that have been debugged and tested.

The amount of information stripped from the symbol table can be controlled by using the following options:

–l Strip line number information only; do not strip any symbol table information.

–x Do not strip static or external symbol information.

–r Reset the relocation indices into the symbol table.

–V Version of *m32strip* command executing.

If there are any relocation entries in the object file and any symbol table information is to be stripped, *m32strip* will complain and terminate without stripping *file-name* unless the **–r** flag is used.

The purpose of this command is to reduce the file storage overhead taken by the object file.

FILES

/usr/tmp/m32str??????

SEE ALSO

m32as(1), m32cc(1), m32ld(1).

DIAGNOSTICS

"m32strip: name: cannot open"
if *name* cannot be read.

"m32strip: name: bad magic"
if *name* is not a a *WE-32001 processor module object file.*

"m32strip: name: relocation entries present; cannot strip"
if *name* contains relocation entries, the **-r** flag not used and any symbol table information was to be stripped.

NAME

proof – troff output simulator for DMD

SYNOPSIS

proof [**–s** ¦ **–w**] file
unproof

DESCRIPTION

The *proof* utility reads *troff* intermediate language output from *file* (standard input default) and displays a simulation of the resulting pages on the screen. The amount of text displayed depends on the mode (which can be set from the command line):

–s *scaled* mode means that each page of text will be compressed into the size of the layer (the default). You may not be able to read it, but you can see what it looks like.

–w *window* mode displays text at the correct size but only what the layer can hold.

Fonts are loaded as required. If the requested font has not yet been scan-converted, the default DMD font is used and the font name of the missing font is noted in the file **tmp/.missing**.

After a layer's worth of text is displayed, *proof* pauses for a command. Typing carriage return turns the page, **q** quits but leaves a "*proof* terminal" (with its loaded fonts), **x** exits and restarts the regular terminal program, **p** *n* sets the next page to be printed to *n*, **s** sets scaled mode, and **w** sets window mode. The page number should be a number or **$** (the last page). Missing numbers go to page 1, out of range numbers go to the next page (as normal). Commands may come from either the host or the keyboard or be selected from the button 3 menu. Keyboard entries other than the legal commands are ignored by *proof*.

The default exit from *proof* leaves a "*proof* terminal" program in the DMD which can be used by later invocations of *proof* without reloading the program and fonts.

Proof interprets lines, splines, circles, ellipses, and arcs directly and without regard to the prevailing point size.

The *Unproof* program restores a "*proof* terminal" to its normal state (equivalent to typing **x** above).

FILES

$DMD/font/*	DMD fonts
$DMD/font/.missing	list of referenced but unconverted fonts

SEE ALSO

troff(1) in the *UNIX System V DOCUMENTER'S WORKBENCH Introduction and Reference Manual.*
UNIX System V DOCUMENTER'S WORKBENCH Software Text Formatters Reference.

BUGS

There is maximum limit to the number of pages (224) that *proof* can handle. If more than the maximum is specified, *proof* will simply print page 224. If the input file does not contain 224 pages or a non-existent page is specified, *proof* will leave the "*proof* terminal" program and return to the UNIX system command mode.

There is a maximum number of fonts (currently 50) that can be loaded at once. Since each (size, typeface) tuple and each special character is a font, this limit can be exceeded in printing highly theoretical papers.

Characters are usually positioned by words, not individually. Character sets that differ from *troff's* idea of their width will result in somewhat ragged copy.

NAME

relogin – rename login entry to show current layer

SYNOPSIS

relogin [**–**]

DESCRIPTION

The *relogin* utility changes your *utmp* entry to the name of the current DMD layer [for *write*(1) and *who*(1) purposes]. It may only be invoked under *layers*(1).

The optional flag suppresses error messages.

Relogin is invoked automatically by *layers*(1) to set the *utmp* entry to the first layer created upon startup, and to reset the *utmp* entry to the real line on termination. It may be invoked by a user when a different DMD layer has been chosen for receiving *write*(1) messages.

FILES

/etc/utmp Data base of people versus terminals.

SEE ALSO

layers(1).
mesg(1), who(1), write(1) in the *UNIX System V User Reference Manual*.
utmp(4) in the *UNIX System V User Reference Manual*.

DIAGNOSTICS

Self explanatory.

NAME

screendump – printer support for DMD 5620 terminals

SYNOPSIS

screendump

DESCRIPTION

The *screendump* utility is a terminal based program running under *jx* that provides support for bitmap output. The *screendump* utility allows the user to select arbitrary screen rectangles for minimal processing and output to either the UNIX system or to the printer port on the 5620 DMD.

Screendump works strictly on screen images, what you see is what you get. The *screendump* layer displays what looks to be an enlarged printer icon. By positioning your mouse over this layer and pressing either button 2 or button 3, a menu appears. The button 3 menu contains entries for bitmap selection and system operation. You can: select a layer sweep a rectangle select the entire screen exit

The button 2 menu contains entries that operate on the selected bitmap. You can: flip the stipple reverse the video run or stop the layer process write the bits to the UNIX system print the bits (on the 5620 DMD output port).

DMD Printer Port

Currently, only support for a C.itoh printer is supplied via the DMD output port. If a printer is connected to this port, the following settings are required:

SW1 2,5,6,7 closed

SW2 7 closed

SW21 all open

SW22 3 closed

SW23 1,5 closed

SW24 1,4,5,8 closed

If the printer is connected to the 3B2 computer system printer, the following settings are required:

SW1 2,6 closed

SW2 7 closed

SW21 4,8 closed

SW22 all opened

SW23 1,5 closed

SW24 2,4,5,8 closed

The *screendump* utility runs only under *layers*(1).

SEE ALSO

jx(1), layers(1), lpg(1).

NAME

sysmon – monitor system activity

SYNOPSIS

sysmon [**–s**] [*seconds*]

DESCRIPTION

The *sysmon* command monitors UNIX system activity on the user's system. Currently, it only runs in a layered environment of the type provided by the 5620 DMD terminal. At the top of the layer is an undulating bar with four different textured portions (from left to right):

solid texture	represents user time
coarse texture	represents kernel time
fine texture	represents wait time
inverse solid texture	represents idle time

Directly below the bar to the left is the time-of-day. The number following the time-of-day is the number of jobs ready to be run. The last number is the increase or decrease in the job's number since the last time it was printed. When new mail arrives, the login id of the sender is printed to the right of these numbers and the keyboard bell rings.

OPTIONS

Presently there are 2 options:

–s prints subject line of mail along with sender id

seconds number of seconds to sleep between updates; default is 3.

FILES

/usr/include/sys/sysinfo.h

SEE ALSO

time(2), mailx(1) in the *UNIX System V User Reference Manual.*

NAME

tek4014 – Tektronix 4014 emulator

SYNOPSIS

tek4014 [strap options] [**–u**]

DESCRIPTION

The *tek4014* utility emulates a Tektronix 4014 computer display terminal. It runs only in *layers* [see *layers*(1)] on the DMD. Facilities for selecting strap options, as well as a variety of choices for portraying "grey levels."

STRAP OPTIONS

Strap	Option	Effect
LFeffect	**–l***	Default: LF causes LF only; **–l**: LF causes LF and CR.
CReffect	**–c***	Default: CR causes CR only; **–c**: CR causes CR and LF.
DELimpliesLOY	**–d**	Default: DEL is legal LOY. **–d**: ESC ? also recognized as DEL.
GINcount	**–e**, -g†	Default: GIN mode sends four position characters, and a CR. **–e**: Send a CR and an EOT. **–g**: Send only the GIN characters; no CR, no EOT. This is needed for *ged*(1).
DIMallowed‡	**–u**	Default: ALLOWED. The screen will dim after 90 seconds idle in ALPHA mode. Recover either by typing a character, or by selecting 'shift' on the button 2 menu. **–u**: The screen never dims.

*NOTE: Only one of **–l** and **–c** may be selected.

†NOTE: Only one of **–e** and **–g** may be selected.

‡NOTE: This is not a true 4014 strap, but is included to make the emulator more usable.

Initialization A layer should be created full screen width, and three fifths of the terminal height for maximal Tektronix compatibility. Distortion of the image will occur otherwise. Type '*tek4014* [options]', and the emulator will be downloaded into the layer.

MOUSE

The button 2 menu shows what option you can select; i.e., it displays what the options are NOT. The following can be selected:

line/local	-	select line or local mode
font	-	select font size
dither/random§	-	
smear/focus§	-	
flash/noflash	-	briefly highlight characters as they are drawn.

§NOTE: The two options above relate to the way that point-plotted images are handled: either by dithering process to get grey levels, or by a coin-flip method; and either by considering 100% to be a 2x2 cell of pixels, or a 1x1. Since a DMD does not have grey levels or the resolution of a real 4014, some fiddling with these may be necessary to produce the best mimicry of a 4014. They start in dither/smear, which produces the best results in most cases.

Button 1 Button 1 is used to select the cross hairs when in GIN mode. Moving the mouse moves the cross hair position.

SEE ALSO

layers(1).
ged(1), graphics(1) in the *UNIX System V User Reference Manual.*
Tektronix 4014 Operator's Manual.

NAME

twid – an interactive drawing program

SYNOPSIS

twid

DESCRIPTION

The *twid* utility is an interactive drawing program that runs under *layers*(1). It allows the user to draw and "paint" on the 5620 DMD terminal using the mouse.

MOUSE BUTTONS

Buttons one and two are used for drawing. The modes they draw in can be changed by selecting an option on the **buttons** menu (see **buttons** below). Button three raises the Supermenu (a menu with menus as selections).

MENU OPTIONS

The *twid* utility has six functional menus; all are options on the Supermenu and each is described below:

style This functional menu permits the user to choose from a variety of drawing tools.

texture This functional menu allows the user to use a group of textures as "paint" and also allows the user to create their own textures.

brush This functional menu allows the user to select from a variety of different brush sizes and also lets the users create their own brush sizes.

buttons This functional menu lets the user determine the modes (**OR**, **XOR**, **CLR**, **STORE**; these correspond to the logical functions or, exclusive or, clear and copy) that buttons one and two possess when used for drawing.

copy This functional menu permits the user to copy and rotate portions of the layer.

unix This functional menu allows the user to write and read bitmaps to and from files stored previously by *twid* on the host machine. This menu also allows the user to exit the *twid* program and return to *layers(1)*.

clear all This option allows the user to sweep out a rectangle and clear portions of the *twid* layer.

FILES

$DMD/lib/twid.m

SEE ALSO

layers(1).

NAME

xtd – extract and print xt driver link structure

SYNOPSIS

xtd [– flags]

DESCRIPTION

The *xtd* command is a debugging tool for the *xt* driver. It performs an XTIOCDATA *ioctl*(2) call on its standard input file to extract the *Link* data structure for the attached group of channels. This call will fail if data extraction has not been configured in the driver or the standard input is not attached to an *xt* channel. The data is printed one item per line on the standard output. The output should probably be formatted via *pr -4*.

The optional flags affect output as follows:

–n (Where **n** is a number in the range 0 to 7) Includes **n** in the list of *channels* to be printed. The default prints all channels, whereas the occurrence of one or more of these channel numbers implies a subset.

–d This flag enables the printout even if the driver is not configured for data extraction (for debugging purposes, only.)

–f Causes a *formfeed* character to be output at the end for the benefit of page display programs.

SEE ALSO

ioctl(2), xt(4), xts(1), xtt(1).
pr(1) in the *UNIX System V User Reference Manual.*

NAME

xts – extract and print xt driver statistics

SYNOPSIS

xts [**–f**]

DESCRIPTION

The *xts* utility is a debugging tool for the *xt* driver. It performs an XTIOCSTATS *ioctl*(2) call on its standard input file to extract the accumulated statistics for the attached group of channels. This call will fail if statistics have not been configured in the driver or the standard input is not attached to an *xt* channel. The statistics are printed one item per line on the standard output.

The optional flag causes a *formfeed* character to be output at the end for the benefit of page display programs.

SEE ALSO

xt(4), xtd(1), xtt(1), ioctl(2).

NAME

xtt – extract and print xt driver packet traces

SYNOPSIS

xtt [**–f**] [**–o**]

DESCRIPTION

The *xtt* utility is a debugging tool for the *xt* driver. It performs an XTIOCTRACE *ioctl*(2) call on its standard input file to turn on tracing and extract the circular packet trace buffer for the attached group of channels. This call will fail if tracing has not been configured in the driver, or the standard input is not attached to an *xt* channel. The packets are printed on the standard output.

The optional flags are:

–f causes a *formfeed* character to be output at the end for the benefit of page display programs.

–o turns off further driver tracing.

SEE ALSO

ioctl(2), xt(4), xtd(1), xts(1).

CONTENTS

NAME

addr – return the Word address of a Point in a Bitmap

SYNOPSIS

Word *addr (b, p)

Bitmap *b;
Point p;

DESCRIPTION

The *addr* function returns the address of the Word containing the bit corresponding to the Point *p* in the Bitmap *b*.

EXAMPLE

The following subroutine can be used to determine whether a Point *p* in a Bitmap *b* is on or off (returning 1 or 0, respectively).

```
#include <dmd.h>

pixel (b, p)
Bitmap *b;
Point p;
{
        Word *w;
        unsigned int bit;

        w = addr (b, p);
        bit = FIRSTBIT >> (p.x%WORDSIZE);
        return (*w&bit)==bit;
}
```

This routine is implemented differently in *rol*(3L).

SEE ALSO

rol(3L).

NAME

agent – DMD agent subroutine

SYNOPSIS

#include <jioctl.h>

agent(ap)
struct bagent *ap;

DESCRIPTION

The *agent* utility performs user-defined functions on the DMD terminal. **Agent** is passed a pointer to the structure *bagent* defined in <**jioctl.h**> as follows:

```
struct    bagent {
    int    size;   /* size of src in & dest out */
    char   *src;   /* the source byte string */
    char   *dest;  /* the destination byte string */
};
```

Src points to the input byte string. *Agent* performs DMD functions on the information in the string and returns information to the host in the destination byte string. Upon return, **size** MUST BE set to the length of the **dest** string.

A default agent routine has been supplied with *layers* and *layersys* which allows a host program to do all mouse button 3 functions. They are accessed from routines referenced in *hagent*(3).

SEE ALSO

hagent(3H), jagent(7H).

NAME

alloc, free – memory allocator

SYNOPSIS

char *alloc (nbytes)
unsigned nbytes;

void free (s)
char *s;

DESCRIPTION

The *alloc* function corresponds to the standard C function *malloc.* **Alloc** either returns a pointer to a block of *nbytes* contiguous bytes of storage or a 0 (NULL) if unavailable. The storage is aligned on 4-byte boundaries. Unlike *malloc*, *alloc* clears the storage to zeros.

The *free* function frees storage allocated by *alloc.*

If a program does not free the memory allocated or if its layer is deleted, *layersys* automatically frees the memory when the layer is deleted. However, it is recommended that a program free its allocated memory when the storage is no longer needed so that other processes operating under *layers* will be able to use it.

EXAMPLE

The following example shows the use of *alloc* and *free* in dynamically allocating memory for a *Point.*

```
#include <dmd.h>

Point *p;

main()
{
    p = (Point *) alloc (sizeof(Point));

    .
    .
    .

    free (p);
}
```

SEE ALSO

balloc(3R), gcalloc(3R).

NAME

atoi – convert string to integer

SYNOPSIS

int atoi (s)

char *s;

DESCRIPTION

The *atoi* function converts a decimal string *s* to an integer. Leading white-space characters, overflow conditions, and conversion stops at the first non-digit encountered are ignored.

EXAMPLE

The following routine will read in from the keyboard a number typed by the user terminated by a carriage return. This routine handles backspace characters and also assumes that the *KBD* resource has been previously requested.

```
#include <dmd.h>

getnum()
{
        char unum[100];
        int i;

        for( i=0; i<99; i++){
           wait(KBD);
           unum[i] = kbdchar();
           if ( unum[i] == '\r' )
              break;
           if ( unum[i] == '\b' && i>0 )
              i -= 2;
        }
        unum[i+1] = '\0';
        return atoi (unum);
}
```

SEE ALSO

itox(3R).

NAME

balloc – bitmap allocator

SYNOPSIS

Bitmap *balloc(r)
Rectangle r;

void bfree(b)
Bitmap *b;

DESCRIPTION

The *balloc* function either returns a pointer to a Bitmap large enough to contain the Rectangle *r*, or a 0 (NULL) for failure. The coordinate system inside the Bitmap is set by *r*. The origin and corner of the Bitmap are those of *r* which must itself be in screen coordinates.

The *bfree* function frees the storage associated with a Bitmap allocated by *balloc*.

If a program does not free the memory allocated or if its layer is deleted, *layersys* automatically frees the memory. However, it is recommended that a program free its allocated memory when the storage is no longer needed so that other processes operating under *layers* will be able to use it.

EXAMPLE

The following example shows the use of *balloc* and *bfree* in dynamically allocating memory for a Bitmap.

```
#include <dmd.h>

Bitmap *b;

main()
{
    b = balloc (Rect (0, 0, 48, 48));

    .
    .
    .

    bfree (b);
}
```

SEE ALSO

alloc(3R), gcalloc(3R).

NAME

bitblt – bit-block transfer

SYNOPSIS

void bitblt(sb, r, db, p, f)
Bitmap *sb, *db;
Rectangle r;
Point p;
Code f;

DESCRIPTION

The *bitblt* function copies the data from Rectangle *r* in Bitmap *sb* to the congruent Rectangle with origin *p* in Bitmap *db*. The nature of the copy is specified by the function Code *f*.

The source and destination Bitmaps may be the same or different and the source and destination Rectangles may even overlap; *bitblt* always does the assignments in the correct order.

EXAMPLE

In order to scroll a Rectangle *r* in a Bitmap *b* by *n* pixels, the following code could be used.

```
#include <dmd.h>

scroll(b, r, n)
Bitmap *b;
Rectangle r;
int n;
{
        Rectangle s;

        s = r;
        s.origin.y += n; /* scroll up */
        bitblt (b, s, b, r.origin, F_STORE);
        s.origin.y = r.corner.y - n; /* clear bottom */
        rectf (b, s, F_CLR);
}
```

SEE ALSO

structures(3R), rectf(3).

NAME

topbits, botbits – bit masking arrays

SYNOPSIS

long topbits[ ]

long botbits[ ]

DESCRIPTION

The array *topbits*[*i*] will reference a long in which the top *i* bits of the long are on and the remaining low bits are off.

The array *botbits*[*i*] will reference a long in which the low *i* bits of the long are on and the remaining high bits are off.

EXAMPLE

For example, *topbits*[4] will equal 0xf0000000, and *botbits*[4] will equal 0x0000000f.

NAME

bttn[123], button[123], bttns – button state

SYNOPSIS

```
int bttn1( ), bttn2( ), bttn3( )
int bttn12( ), bttn13( ), bttn23( ), bttn123( )

int button1( ), button2( ), button3( )
int button12( ), button13( ), button23( ), button123( )

int bttn(b)

int button(b)

void bttns(updown)

int b;
int updown;
```

DESCRIPTION

The functions *bttn1*, *bttn2*, and *bttn3* return the state of the associated mouse button. They return a non-zero if the button is depressed; 0 if not.

The *bttn12* function and the other multi-button functions return a Boolean OR of their states; e.g., true if either button 1 or button 2 is depressed (as opposed to button 1 and button 2).

The *bttn* function takes as an argument the button number 1, 2, or 3 and returns the state of the button.

The *button*[*x*] routines operate in the same manner as each of the *bttn*[*x*] routines except that the *button*[*x*] routines clip to the process layer. This means that the process must be current, the mouse must be owned and inside the process layer, and the proper buttons must be pressed.

Note: It is strongly recommended to use the *button* routines instead of the *bttn* routines.

The *button* function takes as an argument the button number 1, 2, or 3 and returns the state of the button. It also clips to the layer.

The *bttns* function is used to determine when the mouse state changes. When *bttns* is called, it "busy loops"; not returning and not releasing the CPU until the mouse state changes. If *updown* is 0, *bttns* "busy loops" until all buttons are released. If *updown* is 1, *bttns* "busy loops" until any button is depressed. If *updown* is not 0 or 1, *bttns* returns immediately.

EXAMPLE

The following code segment could be written to "doodle" in a layer.

```
#include <dmd.h>

main()
{
    request (MOUSE);
    for (;;){
        wait (MOUSE);
        if (bttn3())
            break;
      if (button1())
            point (&display, mouse.xy,
                  F_STORE);
    }
}
```

NAME

canon – create canonical form of a Rectangle from two Points

SYNOPSIS

Rectangle canon(p1, p2)

Point p1, p2;

DESCRIPTION

The *canon* function creates a Rectangle from two Points such that:

r.origin.x equals the minimum of *p1.x* and *p2.x*
r.origin.y equals the minimum of *p1.y* and *p2.y*
r.corner.x equals the maximum of *p1.x* and *p2.x*
r.corner.y equals the maximum of *p1.y* and *p2.y*

In other words, given the two Points, *canon* creates a Rectangle such that the origin of the Rectangle is the top left Point and the corner of the Rectangle is the bottom right Point.

EXAMPLE

Each of the following cases will yield the Rectangle.

```
{ 0, 0, 32, 32}

canon( Pt(0, 32), Pt(32, 0) )

canon( Pt(32, 32), Pt(0, 0) )

canon( Pt(32, 0), Pt(0, 32) )
```

NAME

circle, disc, discture, arc – circle routines

SYNOPSIS

```
void circle(b, p, r, f)

void disc(b, p, r, f)

void discture(b, p, r, t, f)

void arc(b, p, p1, p2, f)

Bitmap *b;
Point p, p1, p2;
int r;
Texture *t;
Code f;
```

DESCRIPTION

The *circle* function draws the best approximate circle of radius *r* centered at Point *p* in the Bitmap *b* with Code *f*. The circle is guaranteed to be symmetrical about the horizontal, vertical, and diagonal axes.

The *disc* function draws a disc of radius *r* centered at Point *p* in the Bitmap *b* with Code *f*. A disc is a circle which has been completely filled.

The *discture* function draws a disc of radius *r* centered at Point *p* in the Bitmap *b* using the Texture *t* with Code *f*. The *discture* function is similar to the *disc* function except it allows one to specify a pattern to fill the disc.

The *arc* function draws a circular arc centered on *p*, traveling counterclockwise from *p1* to the point on the circle closest to *p2*.

EXAMPLE

The following routine draws a "smiley face" in a Bitmap *b* with

center *c* and radius *r* with nose Texture *t* using Code *f*.

```
#include <dmd.h>

smiley(b, c, rad, t, f)
Bitmap  *b;
Point    c;
int      rad;
Texture *t;
Code     f;
{
    int mino, e; /* offsets for placing */
                 /* eyes, nose and mouth */
    int enrad; /* radius of eyes and nose */

    mino = rad/2;
    e = rad/3;
    enrad = e/3;
    circle (b, c, rad, f); /* face outline */
    disc (b, Pt(c.x-e, c.y-mino),
         enrad, f);        /* left eye */
    disc (b, Pt(c.x+e, c.y-mino),
         enrad, f);        /* right eye */
    discture (b, c, enrad, t, f); /* nose */
    arc (b, c, Pt(c.x-mino,
         c.y+mino), Pt(c.x+mino,
         c.y+mino),f);     /* mouth */
}
```

SEE ALSO

ellipse(3L), jcircle(3R), jellipse(3L), texture(3R).

NAME

isalpha, isupper, islower, isdigit, isxdigit, isalnum, isspace, ispunct, isprint, isgraph, iscntrl, isascii – classify characters

SYNOPSIS

#include <ctype.h>

int isalpha (c)
int c;

DESCRIPTION

These functions classify character-coded integer values by table lookup. Each is a predicate returning non-zero for true, zero for false. The function *isascii* is defined on all integer values; the rest are defined only where *isascii* is true and on the single non-ASCII value –1.

isalpha	*c* is a letter.
isupper	*c* is an upper-case letter.
islower	*c* is a lower-case letter.
isdigit	*c* is a digit [0-9].
isxdigit	*c* is a hexadecimal digit [0-9], [A-F] or [a-f].
isalnum	*c* is an alphanumeric (letter or digit).
isspace	*c* is a space, tab, carriage return, new-line, vertical tab, or form-feed.
ispunct	*c* is a punctuation character (neither control nor alphanumeric).
isprint	*c* is a printing character, code 040 (space) through 0176 (tilde).

isgraph *c* is a printing character, like *isprint* except false for space.

iscntrl *c* is a delete character (0177) or an ordinary control character (less than 040).

isascii *c* is an ASCII character, code less than 0200.

If the argument to any of these functions is not in the domain of the function, the result is undefined.

These are identical to the UNIX system "libc" functions.

NAME

cursinhibit, cursallow, cursswitch, cursset – control cursor

SYNOPSIS

```
void cursallow( )

void cursinhibit( )

Texture16 *cursswitch(t)

void cursset(p)

Texture16 *t;
Point p;

Texture16 C_target, C_crosshair, C_sweep, C_confirm;

Texture16 C_clock, C_cup, C_deadmouse, C_skull, C_move;
```

DESCRIPTION

The *cursinhibit* function turns off the interrupt-time cursor tracking (the drawing of the cursor on the screen), but the mouse coordinates are still kept current and available in the global structure *mouse*.

The *cursallow* function enables interrupt-time cursor tracking.

The functions *cursallow* and *cursinhibit* stack. To enable cursor tracking after two calls to *cursinhibit*, two calls to *cursallow* are required.

The *cursswitch* function changes the mouse cursor to the Texture16 specified by *t*. If *t* is 0, the cursor is restored to the default arrow. The *cursswitch* function returns the previous value of the cursor, and the argument of the previous call to *cursswitch*.

The *cursset* function moves the mouse cursor from the current screen position to the new screen position at Point *p*. The *cursset* function also gives the ownership of the mouse to the calling process if it is running under *layers*.

Each Texture16 listed above is a cursor that is in "ROM" and is accessible by using the names above.

EXAMPLE

The following example divides a layer into four Rectangles. Based on which Rectangle the mouse is in, this program either switches the cursor to the default arrow, switches the cursor to the AT&T

Logo, inhibits the cursor, or sets the cursor to *Drect.origin*.

```
#include <dmd.h>
Texture16 att = {
        0x07E0, 0x1F08, 0x0000, 0x7FFE,
        0x3FC2, 0x0000, 0xFFFF, 0x7FC1,
        0x0000, 0xFFFF, 0x1F01, 0x0000,
        0x7FFE, 0x0000, 0x1008, 0x07E0,
};

main()
{
    Point o, p;
    Rectangle tl, tr, bl, br;
    int lastr = 0;

    o = div (sub (Drect.corner,
            Drect.origin), 2);
    tl.origin = tr.origin = bl.origin
             = br.origin = Drect.origin;
    tr.origin.x += o.x;
    bl.origin.y += o.y;
    br.origin = add (br.origin, o);
    tl.corner = add (tl.origin, o);
    tr.corner = add (tr.origin, o);
    bl.corner = add (bl.origin, o);
    br.corner = add (br.origin, o);
    request (MOUSE|KBD);
    while( kbdchar() == -1 ){
        wait (MOUSE);
        p = mouse.xy;
        if( ptinrect (p, tl) && lastr!=1 ){
          if( lastr==3) cursallow ();
          lastr = 1;
          cursswitch(0);
        } else if( ptinrect (p, tr) &&
                  lastr!=2 ){
                if( lastr==3) cursallow ();
                lastr = 2;
                cursswitch (&att);
```

```
                } else if( ptinrect (p, bl) &&
                          lastr!=3 ){
                        lastr = 3;
                        cursinhibit ();
                } else if( ptinrect (p, br) &&
                          lastr!=4 ){
                        if( lastr==3) cursallow ();
                        lastr = 4;
                        cursset (Drect.origin);
                }
        }
}
```

SEE ALSO

structures(3R).

NAME

ellipse, eldisc, elarc – draw an ellipse

SYNOPSIS

void ellipse(bp, p, a, b, f)

void eldisc(bp, p, a, b, f)

void eldiscture(bp, p, a, b, t, f)

void elarc(bp, p, a, b, p1, p2, f)

```
Bitmap *bp;
Point p, p1, p2;
int a, b;
Texture *t;
Code f;
```

DESCRIPTION

The *ellipse* function draws an ellipse centered at *p* with horizontal semi-axis *a* and vertical semi-axis *b* in Bitmap *bp* with Code *f*.

The *eldisc* function draws an elliptical disc centered at *p* with horizontal semi-axis *a* and vertical semi-axis *b* in Bitmap *bp* with Code *f*.

The *eldiscture* function draws an elliptical disc centered at *p* with horizontal semi-axis *a* and vertical semi-axis *b* in Bitmap *bp* using Texture *t* with Code *f*.

The *elarc* function draws the corresponding elliptical arc, traveling counterclockwise from the ellipse point closest to *p1* to the point closest to *p2*. Note: Differences exist between the calling conventions for *arc* and *elarc*.

EXAMPLE

The following routine can be used to allow a user to sweep out an ellipse by holding button 1 down. When button 1 is released, the ellipse is filled using the eldisc routine.

```
#include <dmd.h>

sweep_eldisc()
{
    Point p, c;
    int a, b;
```

```
        while (!button1() )
            wait (MOUSE);
        c = p = mouse.xy;
        while (button1())
            if (!eqpt(p, mouse.xy)){
                if( !eqpt( p, c))  /* undraw old ellipse */
                    ellipse (&display, c, a, b, F_XOR);
                p = mouse.xy;
                a = abs (p.x - c.x);
                b = abs (p.y - c.y);
                ellipse (&display, c, a, b, F_XOR);
            }
        ellipse (&display, c, a, b, F_XOR);
        eldisc  (&display, c, a, b, F_XOR);
    }
```

SEE ALSO

circle(3L), jcircle(3L), jellipse(3L).

NAME

eqpt, eqrect – compare for equality

SYNOPSIS

int eqpt (p, q)
Point p, q;

int eqrect (r, s)
Rectangle r, s;

DESCRIPTION

The *eqpt* function compares two Points and returns a 1 if the Points are equal or a 0 if they are unequal. Two Points are equal if the corresponding coordinates *x* and *y* are equal.

The *eqrect* function compares two Rectangles and returns a 1 if the Points are equal or a 0 if they are unequal. Two Rectangles are equal if all four corresponding coordinates are equal.

EXAMPLE

The *eqmouse* function determines if the current mouse coordinate equals *p*. The *eqDrect* function determines if the Rectangle passed equals *Drect*.

```
#include <dmd.h>

eqmouse(p)
Point p;
{
        return eqpt (mouse.xy, p);
}

eqDrect(r)
Rectangle r;
{
        return eqrect (Drect, r);
}
```

SEE ALSO

structures(3R).

NAME

exit – cease execution

SYNOPSIS

void exit( );

DESCRIPTION

The *exit* function terminates a process. In either the *stand-alone* or *layers* world, calling *exit* replaces the running process by the default terminal program. Any associated UNIX system process must be terminated separately, since *exit* is a purely local function. When a process calls *exit*, all local resources (keyboard, mouse, storage, etc.,) are deallocated automatically.

EXAMPLE

```
#include <dmd.h>

main()
{
        char c;

        request (KBD);

        .
        .

        if ( (c = kbdchar()) == 'q')
                exit();
        .
        .

}
```

SEE ALSO

resources(3r), kbdchar(3r)

NAME

gcalloc, gcfree – garbage compacting allocator

SYNOPSIS

char *gcalloc (nbytes, where)
unsigned long nbytes;
long **where;

void gcfree(p)
char *p;

DESCRIPTION

The *gcalloc* function provides a simple garbage compacting allocator. It returns either a pointer to a block of *nbytes* contiguous bytes of storage or a NULL, if unavailable. The storage is not initialized to zeros. The pointer *where* points to the user's data where the location of the block is to be saved. The pointer **where* will be updated after each compaction. That is, if garbage collection occurs, your storage will be moved, and your pointer will be changed to point to the new location of your storage area. Therefore, a program using *gcalloc* should never store the location of gcallocated memory anywhere other than the location handed to the allocator. Typically, this location is contained in a structure such as a *Bitmap*.

The *gcfree* function frees the storage block at *p*.

If a program does not free the memory allocated or if its layer is deleted, *layersys* automatically frees the memory. However, it is recommended that a program free its allocated memory when the storage is no longer needed so that other processes, under *layers* , will be able to use it.

EXAMPLE

These routines could be used for allocating and freeing space used to store Points in a polygon.

```
#include <dmd.h>

Point *poly;

Point *
polyalloc(n)
```

```
{
        return (poly = gcalloc)
                (sizeof(Point)*n, &poly ) );
}

polyfree( )
{
        gcfree (poly);
}
```

SEE ALSO

alloc(3R), balloc(3R).

NAME

infont, getfont, outfont, ffree – read a font from the UNIX system

SYNOPSIS

```
#include <font.h>

Font *infont (inch)
int (*inch)( );

Font *getfont (file)
char *file;

int outfont (f, ouch)
Font *f;
int (*ouch)( );

void ffree (f)
Font *f;
```

DESCRIPTION

The *infont* function creates a Font by reading the byte-wise binary representation returned by successive calls to *inch*. It returns **(Font *)0** on error. The argument routine *inch* argument must return successive bytes of the UNIX system file representation of the font, and [**(int)-1**] at end-of-file.

The *outfont* function calls the routine *ouch* to write successive bytes of the binary representation of Font *f*. The *outfont* function returns a –1 on error, as must the routine *ouch*.

For programs running only under *jx*, *getfont* returns a pointer to a Font read from the named *file*, essentially by calling *infont* with argument routine *getc*. The *getfont* function also returns [**(Font *)0**] on error.

The *ffree* function frees a Font allocated by *infont* or *getfont*.

EXAMPLE

See the example for *string*(3R).

SEE ALSO

structures(3R).

NAME

getrect – get Rectangle swept out by user

SYNOPSIS

Rectangle getrect( );

DESCRIPTION

The *getrect* function prompts the user with a box sweep cursor and waits for a Rectangle to be swept out with "button 3." It returns the screen coordinates of the Rectangle swept out. The box may be partly or wholly outside the process's layer.

BUGS

There is no *getrect* call in the *stand-alone* world.

EXAMPLE

The following routine will permit the user to sweep out a Rectangle. As long as the Rectangle swept out does not have any points in common with *Drect*, the user will be prompted to sweep out another Rectangle. Once the swept out Rectangle has points within *Drect*, the Rectangle will be clipped to *Drect* and returned.

```
#include <dmd.h>

Rectangle
sweep_rect( )
{
        Rectangle r;

        do
                r = getrect( );
        while (!rectclip(&r, Drect));
        return r;
}
```

SEE ALSO

rectclip(3r)

NAME

physical, display, Drect, Jrect, XMAX, YMAX, PtCurrent, mouse – globals describing display and mouse

SYNOPSIS

```
Bitmap physical;

Bitmap display;

Rectangle Drect;

Rectangle Jrect;

Point PtCurrent;

#define XMAX 800
#define YMAX 1024

struct Mouse {
    Point xy;
    Point jxy;
    int buttons;
} mouse;
```

DESCRIPTION

Each global is defined when **dmd.h** is included. Users should not include these definitions in their source code.

The global *physical* is a Bitmap describing the entire screen display in screen coordinates.

The global *display* is the Bitmap describing an individual layer in screen coordinates.

The global *Drect* is a Rectangle defining, in screen coordinates, the display area available to the program. It is not *display.rect*, which includes the border around each layer in *layers*.

The global *Jrect* is the Rectangle { 0, 0, *XMAX*, *YMAX* }

The global *PtCurrent* is the current Point in layer coordinates which the j-routines reference and update.

The values *XMAX* and *YMAX* define the maximum x and y coordinates of the DMD screen.

The global *mouse* is a location containing the current mouse coordinates and button states. The Point *xy* is in screen coordinates;

jxy is in layer coordinates. The *buttons* are most easily interpreted using the button functions.

SEE ALSO

buttons(3R), structures(3R).

NAME

openagent, New, Runlayer, Current, Delete, Top, Bottom, Move, Reshape, Exit, Newlayer, openchan – UNIX system host commands to perform DMD functions

SYNOPSIS

```
int     cntlfd, fd;
int     chan;
int     origin.x, origin.y, corner.x, corner.y;
char    *command;

cntlfd = openagent ( )

chan = New (cntlfd, origin.x, origin.y, corner.x, corner.y)

Runlayer (chan, command)

Current (cntlfd, chan)

Delete (cntlfd, chan)

Top (cntlfd, chan)

Bottom (cntlfd, chan)

Move (cntlfd, chan, origin.x, origin.y)

Reshape (cntlfd, chan, origin.x, origin.y, corner.x,

Exit (cntlfd)

chan = Newlayer (cntlfd, origin.x, origin.y, corner.x, corner.y)

fd = openchan (chan)
```

DESCRIPTION

These routines allow a UNIX system host program to issue all the commands available in the "button 3" menu of *layers*. For those routines that pass layer rectangle coordinates, it should be noted that the minimum layer size is 32 by 32 pixels.

The *openagent* function opens the DMD control channel. Upon successful completion, *openagent* returns a file descriptor that is used as input to all routines except *openchan*. Otherwise, the EOF value is returned.

The *New* function creates a new layer with a separate shell. The *origin.x*, *origin.y*, *corner.x*, and *corner.y* arguments are the coordinates of the layer rectangle. If all the coordinate arguments are 0,

the sweep cursor is displayed and the user is allowed to sweep out the layer's rectangle. The layer appears on top of any overlapping layers. The layer is "not" made current (i.e., the keyboard is not attached to the new layer). Upon successful completion, *New* returns the DMD channel associated with the layer. Otherwise, the EOF value is returned.

The *Runlayer* function runs the argument *command* in the layer associated with the channel argument *chan*. The layer must be created with the *New* function.

The *Current* function makes the layer associated with the channel argument *chan* current (i.e., attached to the keyboard).

The *Delete* function deletes the layer associated with the channel argument *chan* and kills all host processes associated with the layer.

The *Top* function makes the layer associated with the channel argument *chan* appear on top of all overlapping layers.

The *Bottom* function puts the layer associated with the channel argument *chan* under all overlapping layers.

The *Move* function moves the layer associated with the channel argument *chan* from its current screen location to a new screen location at the origin point (*origin.x*, *origin.y*). The size and contents of the layer are maintained.

The *Reshape* function reshapes the layer associated with the channel argument *chan*. The arguments *origin.x*, *origin.y*, *corner.x*, and *corner.y* are the new coordinates of the layer rectangle. If all the coordinate arguments are 0, the sweep cursor is displayed and the user is allowed to sweep out the layer's rectangle. The contents of the layer will not be maintained.

The *Exit* function exits the layer's program and kills all processes associated with layer's.

The *Newlayer* function creates a new layer. The arguments *origin.x*, *origin.y*, *corner.x*, and *corner.y* are the coordinates of the layer rectangle. If all the coordinate arguments are 0, the sweep cursor is displayed and the user is allowed to sweep out the layer's rectangle. The layer appears on top of any overlapping layers. The layer is "not" made current (i.e., the keyboard is not attached

to the new layer). Upon successful completion, *Newlayer* returns the DMD channel associated with the layer. Otherwise, the EOF value is returned.

The *openchan* function opens the channel argument *chan*, which is obtained from the *New* or *Newlayer* function. Upon successful completion, *openchan* returns a file descriptor that can be used as input to *write*(2). Otherwise, the EOF value is returned.

NOTES

The file descriptors obtained from the *openagent* and *openchan* functions may be closed using *close*(2).

RETURN VALUE

Upon successful completion, all routines except *openagent*, *openchan*, *New*, and *Newlayer* return a value of 0. If an error occurs, a value of 1 is returned.

SEE ALSO

jagent(7H), agent(3R).
write(2), close(2) in the *UNIX System V Programmer Reference Manual*.

NAME

MPXINDIRECT – accessing rom routine addresses

SYNOPSIS

#define MPXINDIRECT

#include <dmd.h>

DESCRIPTION

The ROM routines are made accessible by a set of **#define**s in an include file **mpx.h**, which is automatically included when one includes **dmd.h**. The proper number of parameters being passed to the routine is automatically validated by the preprocessor through the use of these **#define**s. However, this will not permit one to access the address of a ROM routine. In order to turn off the parameter validation so that a ROM routine address may be accessed, it is necessary to define *MPXINDIRECT* before including **dmd.h.**

NAME

inset – inset a Rectangle for a border

SYNOPSIS

Rectangle inset (r, n)

Rectangle r;
int n;

DESCRIPTION

The *inset* function returns the Rectangle:

{ *r.origin.x+n*, *r.origin.y+n*, *r.corner.x-n*, *r.corner.y-n* }

EXAMPLE

The following code creates a clear Rectangle *r* with a 2-dot wide border inside *r*:

```
rectf(&display, r, F_STORE);
rectf(&display, inset(r, 2), F_CLR);
```

SEE ALSO

structures(3R), recrf(3r).

NAME

abs, ceil, floor, min, max – integer functions

SYNOPSIS

int abs (b)

short ceil (a, b)

short floor (a, b)

int min (b, c)

int max (b, c)

long a;
int b, c;

DESCRIPTION

The *abs* function returns the absolute value of its integer operand.

The *ceil* function returns the smallest short integer which, when multiplied by *b*, is not less than *a*.

The *floor* function returns the largest short integer which, when multiplied by *b*, is not greater than *a*.

For both *ceil* and *floor*, if *b* is equal to 0 or 1, the long integer *a* is truncated to a short and returned.

The *min* function returns the minimum of the two integers.

The *max* function returns the maximum of the two integers.

CAVEAT

The functions *ceil* and *floor* are not the same as the UNIX system functions of the same name.

EXAMPLE

abs (100) and abs (–100) both equal 100.

ceil (7, 2) and ceil (10, 3) both equal 4.

floor (9, 2) and floor (13, 3) both equal 4.

min (32, 16) equals 16.

max (32, 14) equals 32.

NAME

itox – convert integer to hexadecimal string representation

SYNOPSIS

char *itox (n, s)

unsigned long n; char *s;

DESCRIPTION

The *itox* function returns the hexadecimal string representation of the integer *n*. The argument *s* must point to a buffer of at least 11 bytes to place the string. The return value is *s*.

EXAMPLE

```
#include <dmd.h>

      .
      .
      .

int i=12;
char *r;
char buf[11];

      .
      .
      .

         r = itox (i, buf);
```

Note: *r* points to *buf* which contains the hexadecimal value "0xC".

SEE ALSO

atoi(3L).

NAME

jcircle, jdisc, jarc – draw scaled circle on display

SYNOPSIS

void jcircle(p, r, f)

void jdisc(p, r, f)

void jarc(p, p1, p2, f)

Point p, p1, p2;
int r;
Code f;

DESCRIPTION

The *jcircle* function draws in the Bitmap display the approximate circle of radius *r* centered at *p* with Code *f*.

The *jdisc* function draws in the Bitmap display a disc of radius *r* centered at *p* with Code *f*.

The *jarc* function draws the circular arc centered at *p* counter-clockwise from *p1* to the point on the circle closest to *p2* with Code *f*.

All coordinates and radii are in layer coordinates operating under *layers*. Because the layer is scaled, these routines are actually implemented by calls to the ellipse routines.

EXAMPLE

The following routine draws a row of six circles, a row of six discs, and a row of six arcs, scaled to the shape of the layer.

```
#include <dmd.h>

draw()
{
  Point p;
  int i, r;

  r = 50;
  p.y = 200;
  for (p.x=100; p.x < XMAX-50; p.x+=120)
       jcircle (p, r, F_XOR);
  p.y = 600;
```

```
        for (p.x=100; p.x < XMAX-50; p.x+=120)
             jdisc (p, r, F_XOR);
        p.y = 900;
        for (p.x=100; p.x < XMAX-50; p.x+=120)
             jarc (p, Pt(p.x-r, p.y), Pt(p.x+r,
                  p.y), Pt(p.x+r, p.y), F_XOR);
    }
```

SEE ALSO

circle(3L), ellipse(3L), jellipse(3L), parm(3r).

NAME

jellipse, jeldisc, jelarc – draw ellipse on display

SYNOPSIS

void jellipse(p, a, b, f)

void jeldisc(p, a, b, f)

void jelarc(p, a, b, p1, p2, f)

Point p, p1, p2;
int a, b;
Code f;

DESCRIPTION

The *jellipse* function draws in the Bitmap display an approximate ellipse centered at *p*, with horizontal semi-axis *a* and vertical semi-axis *b* with Code *f*.

The *jeldisc* function draws in the Bitmap display an elliptical disc centered at *p*, with horizontal semi-axis *a* and vertical semi-axis *b* with Code *f*.

The *jelarc* function draws the corresponding elliptical arc, centered at *p*, counterclockwise from the ellipse point closest to *p1* to the ellipse point closest to *p2* with Code *f*.

All coordinates and semi-axes are in layer coordinates.

EXAMPLE

The following routine draws a row of six ellipses, a row of six discs, and a row of six arcs, scaled to the shape of the layer.

```
#include <dmd.h>

draw()
{
  Point p;
  int i, r;

  request (MOUSE|KBD);
  r = 50;
  p.y = 200;
  for ( p.x=100; p.x < XMAX-50; p.x+=120)
       jellipse (p, r, r-25, F_XOR);
```

```
        p.y = 600;
        for ( p.x=100; p.x < XMAX-50; p.x+=120)
             jeldisc (p, r-25, r, F_XOR);
        p.y = 900;
        for (p.x=100; p.x < XMAX-50; p.x+=120)
             jelarc (p, r, r, Pt(p.x-r, p.y),
                     Pt(p.x+r, p.y), F_XOR);
    }
```

SEE ALSO

circle(3L), ellipse(3L), jcircle(3L).

NAME

jmove, jmoveto – move current point on display, relative or absolute

SYNOPSIS

void jmove(p)

void jmoveto(p)

Point p;

DESCRIPTION

The *jmove* function moves the current point by the relative vector *p* which is in layer coordinates.

The *jmoveto* function sets the current point to the absolute location *p* which is in layer coordinates.

EXAMPLE

See the example in *jsegment*(3R).

NAME

jpoint – draw single pixel on display

SYNOPSIS

void jpoint(p, f)

Point p;
Code f;

DESCRIPTION

The *jpoint* function sets the pixel at location *p* (in layer coordinates) in the display Bitmap according to the function Code *f*.

EXAMPLE

The following routine can be used to "doodle" on the screen.

```
#include <dmd.h>

doodle()
{
        request (MOUSE);
        for (wait (MOUSE); !button3( );
            wait (MOUSE)){
             if( button1( ) )
                  jpoint (mouse.jxy, F_STORE);
             if( button2( ) )
                  jpoint (mouse.jxy, F_CLR);
        }
}
```

SEE ALSO

jsegment(3R), jmove(3R), point(3R).

NAME

jrectf – rectangle function on display

SYNOPSIS

void jrectf(r, f)

Rectangle r;
Code f;

DESCRIPTION

The *jrectf* function performs the action specified by the function Code *f* on the Rectangle *r* in the display Bitmap. The routine argument *r* is in layer coordinates.

EXAMPLE

The following routine will "doodle" on the screen using a Rectangle, whose coordinates are scaled to the layer.

```
#include <dmd.h>

rectdoodle()
{
        Rectangle r;
        Point s;

        s.x = 16;
        s.y = 16;
        request (MOUSE);
        for (wait(MOUSE); !button3();
             wait(MOUSE)){
             r.origin = mouse.jxy;
             r.corner = add (r.origin, s);
             if( button1() )
                jrectf (r, F_STORE);
             if( button2() )
                jrectf (r, F_CLR);
        }
}
```

SEE ALSO

rectf(3R).

NAME

jline, jlineto, jsegment – draw line on display

SYNOPSIS

void jline(p, f)

void jlineto(p, f)

void jsegment(p, q, f)

Point p, q;
Code f;

DESCRIPTION

The *jline* function draws a line in the display Bitmap with function Code *f* from the current point (initially (0, 0) in layer coordinates) along the relative vector *p* which is in layer coordinates.

The *jlineto* function draws a line from the current point to the absolute layer coordinate *p*. The *jsegment* function draws a line from the layer coordinate *p* to the layer coordinate *q*.

The line functions *jline*, *jlineto*, and *jsegment* leave the current point at the end of the line.

The *PtCurrent* function is the global used to refer to the current point.

EXAMPLE

One of the following routines can be used to draw boxes on the screen.

```
#include <dmd.h>

box(r)
Rectangle r;
{
    jmoveto (r.origin);
    jlineto (Pt (r.corner.x, r.origin.y), F_XOR);
    jlineto (r.corner, F_XOR);
    jlineto (Pt (r.origin.x, r.corner.y), F_XOR);
    jlineto (r.origin, F_XOR);
}

rbox(r, p)
```

```
Rectangle r;
Point p;
{
   jmove (p);
   jline (Pt (r.corner.x - r.origin.x, 0), F_XOR);
   jline (Pt (0, r.corner.y - r.origin.y), F_XOR);
   jline (Pt (r.origin.x - r.corner.x, 0), F_XOR);
   jline (Pt (0, r.origin.y - r.corner.y), F_XOR);
}

sbox(r)
Rectangle r;
{
   jsegment (r.origin, Pt(r.corner.x,
             r.origin.y), F_XOR);
   jsegment (Pt(r.corner.x, r.origin.y),
             r.corner, F_XOR);
   jsegment (r.corner, Pt(r.origin.x,
             r.corner.y), F_XOR);
   jsegment (Pt(r.origin.x, r.corner.y),
             r.origin, F_XOR);
}
```

SEE ALSO

segment(3R).

NAME

jstring – draw character string on display

SYNOPSIS

Point jstring(s)

char *s;

DESCRIPTION

The *jstring* function draws, in the F_XOR mode, the null-terminated string *s* in the display Bitmap so that the origin of the rectangle enclosing the first character is at the current point. It returns the current point, which is left after the last character of the string, so adjacent calls to *jstring* can appear to concatenate their argument strings on the screen.

EXAMPLE

The following routine shows how *jstring* automatically updates *PtCurrent*.

```
#include <dmd.h>

main()
{
        jmove (Pt(16,16));
        jstring ("I");
        jline (Pt(40,0), F_XOR);
        jstring ("love");
        jline (Pt(40,0), F_XOR);
        jstring ("my");
        jline (Pt(40,0), F_XOR);
        jstring ("DMD");

        request (KBD);
        wait (KBD);
}
```

SEE ALSO

string(3R).

NAME

jtexture – draw Texture in Rectangle on display

SYNOPSIS

void jtexture(r, t, f)

Rectangle r;
Texture *t;
Code f;

DESCRIPTION

The *jtexture* function fills the Rectangle *r* in the display Bitmap with Texture *t* using function Code *f*. The *r* argument represents layer coordinates in which a unit of distance may not represent a screen pixel.

EXAMPLE

The following routine allows one to doodle with a Texture.

```
#include <dmd.h>

main()
{
    Rectangle r;
    Point s;

    s.x = 16;
    s.y = 16;
    request (MOUSE);
    for (wait(MOUSE); !button3();
        wait(MOUSE)){
        r.origin = mouse.jxy;
        r.corner = add (r.origin, s);
        if ( button1() )
            jtexture (r, &T_grey, F_STORE);
            if ( button2() )
                jtexture (r, &T_grey, F_CLR);
    }
}
```

SEE ALSO

texture(3R).

NAME

kbdchar – read character from keyboard

SYNOPSIS

int kbdchar( )

DESCRIPTION

The *kbdchar* function returns the next keyboard character typed to the process. If no characters have been typed, *kbdchar* returns –1. If *KBD* has not been requested, *kbdchar* will always return –1, even if characters have been typed to the process.

EXAMPLE

This code will prevent a program from exiting until a character has been typed on the keyboard.

```
#include <dmd.h>

main( )
{
        request (KBD);

        .
        .
        .

        do
                wait( KBD );
        while( kbdchar ( ) == -1 )
}
```

SEE ALSO

resources(3R).

NAME

menuhit – present user with menu and get selection

SYNOPSIS

int menuhit (m, n)

Menu *m;
int n;

```
typedef struct Menu {
   char   **item;          /*string , ending with 0*/
   short  prevhit;          /*from previous call*/
   short  prevtop;          /*from previous call*/
   char   *(*generator)(); /*used if item == 0*/
} Menu;
```

DESCRIPTION

The *menuhit* function presents the user with a menu specified by the Menu pointer *m* and returns an integer indicating the selection made, or -1 for no selection. The *n* argument specifies which button to use for the interaction: 1, 2 or 3. The *menuhit* function assumes that the button is already depressed when it is called. The user makes a selection by releasing the button when the cursor points to the desired selection. releasing the button outside the menu indicates no selection.

The maximum number of menu items displayed at any one time is sixteen items. When the number of items is 16 or less, all of the items are displayed and are centered one entry per line in the menu. This is the normal menu mode. When there are more than 16 menu items to be displayed, the menu becomes a **scrolling** menu. The left portion of the menu contains a scroll bar which is used for scrolling quickly through the menu selections. The vertical size of the scroll bar is an indication of the size of the user's view of the menu (16 items) relative to the number of selections in the entire menu.

There are two ways to scroll through the menu items: The first is to move the mouse cursor to the left side of the menu into the scroll bar area. By moving the mouse cursor up or down within the scroll bar area the menu items will scroll accordingly. The

second method used to scroll through the menu items is to place the mouse cursor on the top or bottom of the menu list. The menu will scroll up or down by one item at a time if there are additional items to be displayed in that direction.

Menus may be generated dynamically from a program by initializing the *generator* function specified in the Menu structure. If *item* is set to 0 when *menuhit* is called, then the routine specified by *generator* is called with one parameter which is an integer index beginning at 0. The *generator* must return a pointer to a character string containing the text for the menu selection. This *generator* function is called repeatedly with the index increasing by 1 until the *generator* returns a NULL, indicating the end of the menu selections. The *generator* function is called each time the *menuhit* routine is called with item set to 0 or NULL.

Another facility provided by *menuhit* is the notion of a spread character. A spread character is any ASCII character with the high-order bit set. The spread character acts somewhat like a spring pushing against the text and borders within a menu entry. The spread character can be placed at the beginning, middle, or end of the string defining the menu entry. If placed at the beginning of the string, the text in the menu item will be right justified. If placed at the end of the string, the text will be left-justified. If placed in the middle of the string, the text on each side of the spread character will be pushed against the menu border. In each of the cases, the space created by the spread character will be filled in with the ASCII character contained in the spread character. For entries without a spread character, the default is to have the text centered.

Whenever a menu is displayed, the original screen image obscured by the menu is saved in the terminal and then later restored when the menu disappears. If the terminal is out of memory and, therefore, cannot save the screen image, then the menu will be displayed in XOR (exclusive or) mode on top of the existing screen image. Menu items may still be selected in this mode but the items might be hard to read. To remedy this problem, memory must be freed up by either deleting or reshaping *layers* before the menu is displayed.

EXAMPLE

The following example includes both a menu with the spread character and a menu that is dynamically generated.

```
#include <dmd.h>

char *menutext[] = {
        "left\240",             /* space char with
                                high bit set */
        "\256right",            /* . char with high
                                   bit set */
        "middle",
        "left\337right",        /* _ char with high
                                   bit set */
        "a very long string",
        NULL };
Menu menu = { menutext };       /* static menu */

/*      Note the above menu will appear as:

         --------------------
         |left                |
         |.............right|
         |       middle       |
         |left _________right|
         |a very long string|
         --------------------
*/

static char digits[50];
static int index;
#define MAXDIGIT 25

pic9( n )
long n;
{
        if(n/10) pic9 (n/10);
        digits[index++] = "0123456789"[n%10];
}
```

```
char *
decimal(i)
long i;
{
        index = 0;
        pic9 ((long)i);
        digits[index] = '\0';
        return digits;
}
char *
generate(i)
int i;
{
        if( i > MAXDIGIT ) return NULL;
        return decimal (i);
}

Menu menugen = {0, 0, 0, generate};
             /* dynamically generated menu */


main()
{
        int m;

        request (MOUSE);
        while( !bttn3() )
                wait (MOUSE);
        m = menuhit (&menu, 3);
        if ( m == 0 )
                exit();
        while( !bttn2() )
                wait (MOUSE);
        m = menuhit (&menugen, 2);
}
```

SEE ALSO

layers(1).

NAME

muldiv – calculate (a*b)/c accurately

SYNOPSIS

int muldiv (a, b, c)
int a, b, c;

DESCRIPTION

The *muldiv* function is a macro that returns the 32-bit result *(a*b)/c*. *(a*b)* is calculated to 32 bits to mimimize the precision lost. The *muldiv* function is convenient for calculating transformations.

EXAMPLE

To calculate, for example, the mathematical expression:

```
x = x0*cos (d)
```

To calculate a projection, the multiplication must be scaled:

```
x = muldiv (x0, cos(d), 1024)
```

Another example is the code for *jeldisc*.

```
#include <dmd.h>

void
jeldisc(p, a, b, f)
Point p;
int a, b, f;
{
   eldisc (&display, transform(p), muldiv(a,
        Drect.corner.x-Drect.origin.x, XMAX),
      muldiv(b,
        Drect.corner.y-Drect.origin.y, YMAX),
      f);
}
```

SEE ALSO

trig(3L).

NAME

norm, sqrtryz – return norm or coordinate of three-dimensional vector

SYNOPSIS

int norm (x, y, z)

int sqrtryz (r, y, z)

int r, x, y, z;

DESCRIPTION

The *norm* utility returns the norm of the vector (*x*, *y*, *z*). It is defined by the equation:

$$\sqrt{x^2+y^2+z^2}$$

The *sqrtryz* call returns the x-coordinate when passed the norm *r* and *y* and *z* coordinates. It is defined by the equation:

$$\sqrt{r^2-y^2-z^2}$$

NAME

outline – draw the outline of a Rectangle

SYNOPSIS

void outline (r)

Rectangle r;

DESCRIPTION

The *outline* function draws the outline of the Rectangle *r* in the Bitmap *physical* using function Code *F_XOR*. The right and lower edges are decremented by one pixel before the outline is drawn so that abutting Rectangles have no points in common. Note that since *outline* writes to the Bitmap *physical*, the outline will be displayed on the screen without clipping to a layer or placing the outline in an obscured Bitmap.

EXAMPLE

The following code could be used to track a Rectangle with the mouse.

```
#include <dmd.h>
trackRect(tr)
Rectangle tr;
{
        Rectangle r;
        Point     p,q;
        request (MOUSE);
        while (!button1())
              wait(MOUSE);
        q = mouse.xy;
        r = raddp (tr, q);
        outline (r);
```

```
        do {
                wait (MOUSE);
                p = mouse.xy;
                if (!eqpt(p, q)){
                        outline (r);
                        r = raddp (tr, p);
                        outline (r);
                        q = p;
                }
        } while( button1() );
}
```

NAME

Pt – create a Point from two coordinates
Rpt – create a Rectangle from two Points
Rect – create a Rectangle from four coordinates

SYNOPSIS

```
Point Pt (x, y)
int x, y;

Rectangle Rpt (p, q)
Point p, q;

Rectangle Rect (a, b, c, d)
int a, b, c, d;
```

DESCRIPTION

Parms are special routines that are **only** to be used in an argument list to a function.

The *Pt* argument passes a coordinate pair as a Point to a function.

The *Rpt* argument passes two Points as a Rectangle to a function.

The *Rect* argument passes four coordinates (two coordinate pairs) as a Rectangle to a function.

EXAMPLE

The following examples show the use of the functions.

```
#include <dmd.h>

        .
        .
        .

outline (Rpt (add (Drect.origin,
      Pt (2,2)), Drect.corner));

outline (Rect (0,0,32,32));
```

NAME

pfkey – get programmable function key values

SYNOPSIS

int pfkey (keynum, str, maxlen)

int keynum, maxlen;
char *str;

DESCRIPTION

By default, the programmable function keys are automatically expanded and placed on the keyboard queue in their expanded form. If a program performs its own interpretation of the keys, it is necessary to set a bit in the process state variable:

```
P->state &= NOPFEXPAND;
```

The programmable function keys will have values 0x82 through 0x89 for keys f1 through f8, respectively.

The *pfkey* places up to **(maxlen-1)** characters from the given programmable function key *keynum* into the character array **str**.

When *pfkey* encounters a null value or it has processed the **(maxlen-1)**th character, *pfkey* terminates the string with a NULL and returns the length of the string.

EXAMPLE

The following code places the value of function key 1 into *pfval*.

```
#include <dmd.h>

#define MAXPFKEY 80
char pfval[MAXPFKEY];

pfkey (0x82, pfval, MAXPFKEY);
```

NAME

plot – graphics interface subroutines

SYNOPSIS

```
openpl ()

erase ()

label (s)
char *s;

line (x1, y1, x2, y2)
int x1, y1, x2, y2;

circle (x, y, r)
int x, y, r;

arc (x, y, x0, y0, x1, y1)
int x, y, x0, y0, x1, y1;

move (x, y)
int x, y;

cont (x, y)
int x, y;

point (x, y)
int x, y;

linemod (s)
char *s;

space (x0, y0, x1, y1)
int x0, y0, x1, y1;

closepl ()
```

DESCRIPTION

These subroutines generate graphic output in a relatively device-independent manner. *Space* must be used before any other of these functions to declare the amount of space necessary [see *plot*(4)]. *Openpl* must be used before any of the others to open the device for writing. *Closepl* flushes the output.

Circle draws a circle of radius *r* with center at the point *(x, y)*.

Arc draws an arc of a circle with center at the point *(x, y)* between the points *(x0, y0)* and *(x1, y1)*.

String arguments to *label* and *linemod* are terminated by nulls and do not contain new-lines.

See *plot*(4) for a description of the effect of the remaining functions.

The library files listed below provide several flavors of these routines.

FILES

/usr/lib/libplot.a	produces output for *tplot* (1G) filters
/usr/lib/lib300.a	for DASI 300
/usr/lib/lib300s.a	for DASI 300s
/usr/lib/lib450.a	for DASI 450
/usr/lib/lib4014.a	for Tektronix 4014
$DMD/lib/lib5620.a	for dmd 5620

WARNINGS

In order to compile a program containing these functions in *file.c*, it is necessary to use "cc *file.c* –lplot".

In order to execute it, it is necessary to use "a.out | tplot".

The above routines use <**stdio.h**>, which causes them to increase the size of programs (not otherwise using standard I/O) more than might be expected.

SEE ALSO

plot(4), tplot(1G).
graph(1G), stat(1G) in the *UNIX System V User Reference Manual*.

NAME

point – draw a single pixel in a Bitmap

SYNOPSIS

void point (b, p, f)

Bitmap *b;
Point p;
Code f;

DESCRIPTION

The *point* function draws the pixel at Point *p* in the Bitmap *b* according to function Code *f*.

EXAMPLE

The following routine determines the Point at the middle of the layer, draws and then returns it.

```
#include <dmd.h>

Point middle()
{
        Point center, offset;

        offset = div (sub (Drect.corner,
                Drect.origin), 2);
        center = add (Drect.origin, offset);
        point (&display, center, F_XOR);
        return center;
}
```

SEE ALSO

jpoint(3R).

NAME

polyf, ptinpoly – polygon routines

SYNOPSIS

int polyf (bp, poly, np, t, f)

int ptinpoly (pt, poly, np)

Point pt;
Bitmap *bp;
Point poly[ ];
short np;
Texture *t;
Code f;

DESCRIPTION

The *polyf* function is used to fill a closed polygon defined by the *np* Points in the array of Points *poly*. The Points are absolute with respect to the Bitmap *bp*. The polygon is filled with the Texture *t* using Code *f*. The *polyf* call returns 0 if the polygon is filled and -1 if a memory allocation error occurred during processing, in which case the polygon is not filled.

The *ptinpoly* call determines whether the Point *pt* is contained in the polygon defined by *poly* and *np*. The *ptinpoly* function returns 1 if the Point is inside the polygon, 0 if the Point is not inside the polygon, and -1 if a memory allocation error occurs.

The polygon can consist of an arbitrary number of filled and unfilled regions. For example, a donut shape could be formed without affecting the portion of the Bitmap corresponding to the hole of the donut. This permits a preservation of any background information previously placed in the Bitmap. The array of Points *poly* can have an arbitrary number of alternating filled and unfilled regions, with each region delimited by a Point whose x-coordinate has a value of *POLY F* (defined in **dmd.h**). This Point in *poly* is ignored and merely serves as a flag indicating the start of a new region. There is always an assumed line connecting the first and last Point of each region.

EXAMPLE

The following code permits interactive drawing of a polygon with

interior regions, fills it, and then uses *ptinpoly* to determine if the mouse is inside the polygon.

```
#include <dmd.h>

Point npoly[1000];

main()
{
        register int i, j;

        request (MOUSE|KBD);
        for( i = 0; i < 1000;){
                wait (MOUSE);
                if( button1 () ){
                        npoly[i++] = mouse.xy;
                        point (&display, mouse.xy,
                                F_STORE);
                        sleep (10);
                        continue;
                }
                if( button2 () ){
                        npoly[i++].x = POLY_F;
                        continue;
                }
                if( button3 () ) break;
        }
        polyf (&display, npoly, i, &T_black,
                F_STORE);
        while( kbdchar () == -1 ){
                wait (MOUSE);
                if( button1() && ptinpoly
                        (mouse.xy, npoly, i) )
                        ringbell();
        }
}
```

NAME

psendchar – send character to printer port

SYNOPSIS

int psendchar (c)

char c;

DESCRIPTION

The *psendchar* function sends a single byte to the printer output queue. If the queue is full, *psendchar* returns a 0. Otherwise, *psendchar* initiates printing and returns the number of characters on the output queue.

If no printer is attached to the terminal, the characters are lost.

NAME

add, sub, mul, div – arithmetic on Points

SYNOPSIS

Point add(p, q)

Point sub(p, q)

Point mul(p, a)

Point div(p, a)

Point p, q;
int a;

DESCRIPTION

The *add* function returns the Point sum of its arguments:

```
{ p.x+q.x, p.y+q.y }
```

The *sub* function returns the Point difference of its arguments:

```
{ p.x-q.x, p.y-q.y }
```

The *mul* function returns the Point:

```
{ p.x*a, p.y*a }
```

The *div* function returns the Point:

```
{ p.x/a, p.y/a }.
```

EXAMPLE

The following routine returns the center Point of a *layer*.

```
#include <dmd.h>
Point
getcenter()
{
    Point offset;
    offset = div (sub
        (Drect.corner,Drect.origin), 2);
    return add (Drect.origin, offset);
}
```

SEE ALSO

layers(1).

NAME

ptinrect – check for Point inclusion in a Rectangle

SYNOPSIS

int ptinrect(p, r)

Point p;
Rectangle r;

DESCRIPTION

The *ptinrect* function returns 1 if *p* is a Point within Rectangle *r*, otherwise, 0 is returned.

EXAMPLE

The following routine will draw a box in the middle of a layer. Then, when the user hits button 1, the bell will ring. The routine exits when a key is typed.

```
#include <dmd.h>

ringbox()
{
    Point center, offset;
    Rectangle midbox;

    offset = div (sub (Drect.corner,
                Drect.origin), 2);
    center = add (Drect.origin, offset);
    midbox.origin = sub (center, Pt (16, 16));
    midbox.corner = add (center, Pt (16, 16));
    outline (midbox);
    request (MOUSE|KBD);
    while (kbdchar () == -1){
        wait (MOUSE);
        if (ptinrect (mouse.xy, midbox) &&
                button1()) ringbell();
    }
}
```

NAME

qsort – quick sort

SYNOPSIS

void qsort (base, nel, sizeof (*base), compar)

char *base; unsigned int nel;
int (*compar)( );

DESCRIPTION

The *qsort* function is an implementation of the UNIX system quicksort algorithm. It sorts a table of data in place.

The *base* points to the element at the base of the table. The pointer to the base of the table should be of type pointer-to-element, and cast to type pointer-to-character.

The *nel* argument is the number of elements in the table.

The *compar* element is the name of the comparison function which is called with two arguments that point to the elements being compared. It need not compare every byte, so arbitrary data may be contained in the elements in addition to the values being compared. The function must return an integer less than, equal to, or greater than zero, based on whether the first argument is to be considered less than, equal to, or greater than the second.

EXAMPLE

The following routine will accept Points designated by clicking button 1 and then sort the array based on *x* and *y* coordinate values.

```
#include <dmd.h>

Point p[100];

ptcompar(p, q)
Point *p, *q;
{
        if ( p->y < q->y ) return -1;
        if ( p->y == q->y ) {
                if ( p->x == q->x ) return 0;
                if ( p->x < q->x ) return -1;
```

```
                return 1;
        }
        return 1;
}

Point *
sortpoints()
{
        int i;

        request (MOUSE|KBD);
        for( i = 0; i < 100;){
                wait (MOUSE);
                if( button1 () ){
                        p[i++] = mouse.xy;
                        sleep (10);
                        continue;
                }
                if( button3 () ) break;
        }
        qsort ((char *)p, i, sizeof(Point),
                ptcompar);
        return p;
}
```

NAME

rand, srand – simple random-number generator

SYNOPSIS

int rand ( )

void srand (seed)
unsigned seed;

DESCRIPTION

The *rand* function uses a multiplicative congruential random-number *generator* with period 2^{32} that returns successive pseudo-random numbers in the range from 0 to $2^{15}-1$.

The *srand* function can be called at any time to reset the random-number generator to a random starting point. The generator is initially seeded with a value of 1.

NAME

rcvchar – receive character from host

SYNOPSIS

int rcvchar ( )

DESCRIPTION

The *rcvchar* function returns the next character received from the host. If there are no characters available or *RCV* was not requested, *rcvchar* returns -1.

EXAMPLE

The following routine could be used to wait until a character has been received from the host and then return it.

```
#include <dmd.h>

char
gethostchar()
{
        int c;

        request(own()&RCV);
        do
                wait (RCV);
        while ((c = rcvchar()) == -1);
        return (char)c;
}
```

SEE ALSO

resources(3R).

NAME

realtime – wall clock

SYNOPSIS

long realtime ( )

DESCRIPTION

The *realtime* function returns the number of 60 Hz clock ticks since *layers* was booted. There is no *realtime* call in the standalone world.

SEE ALSO

bitblt(3), layers(1).

EXAMPLE

The following code could be used to roughly measure the performance of *bitblt*.

```
#include <dmd.h>

perf_bitblt( sb, r, db, p, f)
Bitmap *sb, *db;
Rectangle r;
Point p;
Code f;
{
        long begin;

        begin = realtime ();
        bitblt (sb, r, db, p, f);
        return (realtime () - begin);
}
```

NAME

raddp, rsubp – arithmetic on Rectangles

SYNOPSIS

Rectangle raddp (r, p)

Rectangle rsubp (r, p)

Rectangle r;
Point p;

DESCRIPTION

The *raddp* function returns the Rectangle defined by [**add(r.origin, p)**, **add(r.corner, p)**].

The *rsubp* function returns the Rectangle defined by [**sub(r.origin, p), sub(r.corner, p)**].

SEE ALSO

structures(3R).

EXAMPLE

The following code will create a box with the mouse coordinates at the center of the box.

```
#include <dmd.h>

Rectangle r;
     .
     .
     .

r = raddp (Rect(-15, -15, 15, 15),
           mouse.xy);

     .
     .
     .
```

NAME

rectclip – clip Rectangle to another Rectangle

SYNOPSIS

int rectclip(rp, s)

Rectangle *rp, s;

DESCRIPTION

The *rectclip* function clips the Rectangle in place pointed to by *rp* so that it is completely contained within the Rectangle *s*. The return value is 1 if any part of **rp* is within *s*. Otherwise, the return value is 0 and **rp* is unchanged.

SEE ALSO

getrect(3R).

EXAMPLE

See the example for *getrect*(3R).

NAME

rectf – perform function on Rectangle in Bitmap

SYNOPSIS

void rectf (b, r, f)

Bitmap *b;
Rectangle r;
Code f;

DESCRIPTION

The *rectf* function performs the action specified by the function Code *f* on the Rectangle *r* within the Bitmap *b*.

SEE ALSO

jrectf(3R).

EXAMPLE

The following routine will "doodle" on the screen using a Rectangle.

```
#include <dmd.h>

rectdoodle()
{
    Rectangle r;
    Point s;

    s.x = 16;
    s.y = 16;
    request (MOUSE);
    for (wait(MOUSE); !button3();
      wait(MOUSE)){
        r.origin = mouse.jxy;
        r.corner = add (r.origin, s);
        if( button1() )
            rectf (&display, r, F_STORE);
        if( button2() )
            rectf (&display, r, F_CLR);
    }
}
```

NAME

rectXrect – do the Rectangles overlap?

SYNOPSIS

int rectXrect(r, s)

Rectangle r, s;

DESCRIPTION

The *rectXrect* function returns 1 if *r* and *s* share any point, otherwise, 0 is returned.

EXAMPLE

The following routine will prompt the user to sweep out two Rectangles. If they intersect, the bell will ring.

```
#include <dmd.h>

main()
{
        Rectangle r, s;

        request (KBD);
        do {
                r = getrect ();
                outline (r);
                s = getrect ();
                outline (s);
                if (rectXrect (r, s))
                        ringbell ();
                outline (r);
                outline (s);
        } while (kbdchar()==-1);
}
```

NAME

request, own, wait, alarm – routines dealing with resources

SYNOPSIS

```
int request(r)

int own( )

int wait(r)

void alarm(t)

int r;
unsigned t;
```

DESCRIPTION

Each of these routines deals with the following DMD resources:

KBD	keyboard
SEND	characters sent from DMD to UNIX system process
MOUSE	mouse
RCV	characters received by DMD from UNIX system process
CPU	DMD cpu
ALARM	pseudo resource

A bit vector indicating which resources are being referenced is composed by using the boolean **or** procedure on the above constants (MOUSE|KBD means you are referring to the mouse and keyboard resources). If the keyboard is not requested, characters typed will be sent to the standard input of the UNIX system process. If the mouse is not requested, mouse events in the process layer will be interpreted by *layersys* rather than passed to the process. SEND and CPU are always implicitly *request*ed.

The *request* routine announces a program's intent to use I/O devices and resources and is usually called once early in the program. The **r** argument is a bit vector indicating which resources are to be used. The routine returns a bit vector indicating the resources that you own.

Note that if a program calls *request* several times, each *request* overrides the previously *request*ed resources. This means that a resource previously *request*ed, but not in the latest call to *request*,

will no longer be owned.

The *own* function returns a bit vector of which I/O resources have data available.

The *wait* function suspends the process, enabling others, until at least one of the *request*ed resources is available. The return value is a bit vector indicating which of the requested resources are available. Processes wishing to give up the processor to enable other processes to run may call *wait***(CPU)**. It will return as soon as all other active processes have had a chance to run. The *wait***(SEND)** call is a no-op but is not guaranteed to wait.

The *alarm* function starts a timer which will "fire" **t** ticks (60ths of a second) in the future. A pseudo-resource **ALARM** can be used to check the status of the timer. Calling *alarm* implicitly *request*s the **ALARM** pseudo-resource. The *own*( ) **&ALARM** call indicates whether the alarm timer has fired.

The *alarm* call does not interfere with *sleep* and vice versa.

EXAMPLE

The following routine shows how one can give up button 3 processing to *layersys*.

```
#include <dmd.h>

int o;
    request (MOUSE);
    o = own();
        if (bttn3()) {
                request (o&~MOUSE);
                wait (CPU);
                request (o);
        }
```

NAME

ringbell – ring the DMD bell

SYNOPSIS

```
void ringbell ( )
```

DESCRIPTION

This function rings the bell on the DMD.

EXAMPLE

See the example in *polygon*(3R).

SEE ALSO

polygon(3R).

NAME

rol, ror – rotate bits

SYNOPSIS

int rol (x, n)

int ror (x, n)

int x, n;

DESCRIPTION

The *rol* function returns *x* logically bit-rotated left by *n*.

The *ror* function returns *x* logically bit-rotated right by *n*.

EXAMPLE

The following subroutine can be used to determine whether a Point *p* in a Bitmap *b* is on or off (returning 1 or 0, respectively).

```
#include <dmd.h>

pixel (b, p)
Bitmap *b;
Point p;
{
        Word *w;

        w = addr (b, p);
        return ( (rol (*w, p.x%WORDSIZE)
                & FIRSTBIT)==FIRSTBIT);
}
```

This routine is implemented differently in *addr*(3R).

SEE ALSO

addr(3R).

NAME

screenswap – swap screen Rectangle and Bitmap

SYNOPSIS

void screenswap (b, r, s)

Bitmap *b;
Rectangle r, s;

DESCRIPTION

The *screenswap* function does an in-place exchange of the screen Rectangle *s* and the Rectangle *r* within the Bitmap *b*. Its action is undefined if *r* and *s* are not congruent. The *s* argument is clipped to the screen, not the layer's Rectangle.

EXAMPLE

The following code fragment is an excerpt from the *menuhit* routine.

```
#include <dmd.h>

Bitmap *b;
Rectangle r;

   b = balloc (r);
   cursinhibit ();
   if (b)
         bitblt (&physical, r, b, r, F_STORE);
   rectf (&physical, r, F_OR);
   cursallow ();

   .
   .

   if (b){
      cursinhibit ();
      screenswap (b, b->rect, b->rect);
      cursallow ();
      bfree (b);
   }
```

The calls to *cursinhibit* and *cursallow* is to avoid any unwanted interaction with the cursor when copying to and from the screen bitmap.

SEE ALSO

menuhit(3R).

NAME

segment – draw a line segment in a Bitmap

SYNOPSIS

void segment (b, p, q, f)

Bitmap *b;
Point p, q;
Code f;

DESCRIPTION

The *segment* utility draws a line segment in Bitmap *b* from Point *p* to Point *q* with function Code *f*.

Like all the other graphics operations, *segment* clips the line so that only the portion of the line intersecting the Bitmap is displayed.

SEE ALSO

jsegment(3R).

EXAMPLE

The following call simply draws a line connecting a layer's origin Point to its corner Point.

```
#include <dmd.h>

        .
        .
        .

segment (&display, Drect.origin,
         Drect.corner, F_XOR);

        .
        .
        .
```

NAME

sendchar, sendnchars – send character(s) to host

SYNOPSIS

```
int sendchar (c)
char c;

void sendnchars (n, p)
int n;
char *p;
```

DESCRIPTION

The *sendchar* function sends a single byte to the host which will normally be read on the standard input of the UNIX system process. The characters are passed to the UNIX system *tty(1)* input routine and will be processed as though they were typed on the keyboard by a user. In *layers, sendchar* always returns a 1. In *stand-alone, sendchar* returns a 0 if the character was placed on the queue. But if the queues are full, the character is not placed on the queue, and the number of characters on the queue is returned.

The *sendnchars* function sends *n* characters pointed to by *p* to the host. *Sendnchars* is obtained from a library for programs running *stand-alone.*

These routines block if the output queue is full.

EXAMPLE

```
#include <dmd.h>

char  c;

      .
      .
      .

      sendchar (c);
```

SEE ALSO

layers(1), resources(3R).
tty(1) in the *UNIX System V User Reference Manual.*

NAME

sleep, nap, sw – suspend program execution

SYNOPSIS

void sleep (nticks)

void nap (nticks)

void sw (run)

int nticks;
int run;

DESCRIPTION

The *nap* function busy loops for *nticks* ticks of the 60 Hz internal clock. To avoid interfering with screen refresh, programs drawing rapidly changing scenes should call *nap* between updates to synchronize the display and memory.

The *sleep* function is identical to *nap* except that it gives up the processor for the interval. Unless using the mouse, a program should call *sleep* in preference to *nap*.

The *sw* function is called by a *layers* process when it is ready to allow another process to execute. *Sw* may be called directly and is also called by *wait* and *sleep*. A process that never calls *sw*, *wait*, or *sleep* can lock out all other DMD processes.

If *run* is 1, the process asks to be re-scheduled and will run again after all other processes have had their turn. If *run* is 0, the process asks to not be run again, until either an *alarm* fires or a character is received from either the keyboard of the UNIX System host.

SEE ALSO

layers(1), resources(3L).

NAME

sqrt – integer square root

SYNOPSIS

int sqrt (x)

int x;

DESCRIPTION

The *sqrt* function returns the 32-bit signed integer closest to the square root of its 32-bit signed argument.

SEE ALSO

norm(3L).

NAME

P->state – Process state variable

SYNOPSIS

#include <dmd.h>

int P->state;

DESCRIPTION

P->state is the state variable for a process running in the DMD. There are two fields in this variable which a program may need to deal with: MOVED, and RESHAPED. These bits are set by *layersys*, but if a program handles the display recovery of being moved or reshaped, it must reset the bits using the following code.

For a process to test whether its *layer* has been MOVED , it must check:

```
( P->state&MOVED )
```

To reset the MOVED condition, it is necessary to execute:

```
P->state &= ~(MOVED|RESHAPED);
```

For a process to test whether its *layer* has been RESHAPED, it must check:

```
( P->state&RESHAPED && !(P->state&MOVED) )
```

To reset the RESHAPED condition, it is necessary to execute:

```
P->state &= ~RESHAPED;
```

The reason for the interaction of the MOVED and RESHAPED bits is purely historical.

SEE ALSO

layers(1), pfkey(3R).

NAME

string, defont – draw string in bitmap

SYNOPSIS

```
#include <font.h>

Point string (ft, s, b, p, f)

Font *ft;
char *s;
Bitmap *b;
Point p;
Code f;

Font defont;
```

DESCRIPTION

The *string* function draws the null-terminated string *s* using characters from Font *ft* in Bitmap *b* at Point *p* with function Code *f*. The return value is the location of the first character after *s*, passed to another call to string. The two strings will be concatenated.

The characters are drawn such that the origin point of the bounding rectangle of a maximum height character lies at *p*. Therefore, a character drawn on the screen at (0, 0) will occupy the upper left-most character position on the screen.

The *string* function draws characters as they are in the font. No special action is taken for control characters such as tabs or newlines.

The global *defont* is predefined and is the name of the standard font (not a pointer to it).

SEE ALSO

jstring(3R), structures(3R).

EXAMPLE

Following is an example for the use of *string*.

Here is a code fragment that prints " hello world" in a large font.

```
#include <dmd.h>
#include <font.h>

        .
        .
        .


Font *f;
Point p;

p = add (Drect.origin, Pt(4,4));
if( (f = getfont
      ("/usr/dmd/font/HI.24")) == 0)
        exit();
string (f, "hello world",
      &display, p, F_XOR);

ffree (f);
```

NAME

strcat, strncat, strcmp, strncmp, strcpy, strncpy, strchr, strrchr – string operations

SYNOPSIS

```
char *strcat (s1, s2)

char *strncat (s1, s2, n)

int strcmp (s1, s2)

int strncmp (s1, s2, n)

char *strcpy (s1, s2)

char *strncpy (s1, s2, n)

char *strchr (s1, c)

char *strrchr (s2, c)

char *s1, *s2;
int c, n;
```

DESCRIPTION

The arguments *s1* and *s2* point to strings (arrays of characters terminated by a null character). The functions *strcat*, *strncat*, *strcpy*, and *strncpy* all alter *s1*. These functions do not check for overflow of the array pointed to by *s1*.

The *strcat* function appends a copy of string *s2* to the end of string *s1*. It appends at most *n* characters. Each returns a pointer to the null-terminated result.

The *strcmp* function compares its arguments and returns an integer less than, equal to, or greater than 0 (whether *s1* is lexicographically less than, equal to, or greater than *s2*). The *strncmp* function makes the same comparison but looks at most *n* characters.

The *strcpy* function copies string *s2* to *s1*, stopping after the null character has been copied. It copies exactly *n* characters, truncating *s2* or adding null characters to *s1*, if necessary. The result will not be null-terminated if the length of *s2* is *n* or more. Each function returns *s1*.

The *strchr* function (*strrchr*) returns a pointer to the first (last) occurrence of character **c** in string **s** or a NULL pointer if **c** does not occur in the string. The null character terminating a string is considered to be part of the string.

SEE ALSO

strlen(3R).

NAME

strlen – length of character string

SYNOPSIS

int strlen (s)
char *s;

DESCRIPTION

The *strlen* function returns the number of characters in string *s* , not including the terminating NULL character.

EXAMPLE

```
#include <dmd.h>

int   len;
char *s = "This is an example";

      .
      .
      .

      len = strlen (s);
```

The length of the string " This is an example" is 18 characters and *len* will then equal 18.

SEE ALSO

stringc(3L).

NAME

Word, Code, Point, Rectangle, Bitmap, Texture, Texture16, Font, Fontchar – DMD Structures

SYNOPSIS

#include dmd.h

#include font.h

DESCRIPTION

In the following summaries, all coordinates are screen or Bitmap coordinates (which are scaled the same) unless specified as layer coordinates. With the exception of *Font* and *Fontchar*, these structure definitions are included in a program by including **dmd.h**. *Font* and *Fontchar* are defined in **font.h**.

Word

```
typedef  int  Word;
```

A *Word* is a 32-bit integer and is the unit of storage used in the graphics software.

Code

```
typedef int Code;
```

Code is the functional constant used in all graphical drawing or copying operations. Possible Codes are:

```
F_STORE     target  =  source
F_OR        target |=  source
F_XOR       target ^=  source
F_CLR       target &= ~source
```

Point

```
typedef struct Point {
   short  x;
   short  y;
} Point;
```

A *Point* is a location in a *Bitmap*, such as the display. The coordinate system has *x* increasing to the right and *y* increasing down.

Rectangle

```
typedef struct Rectangle {
   Point  origin;          /* Upper left */
   Point  corner;          /* Lower right */
} Rectangle;
```

A *Rectangle* is a rectangular area in a *Bitmap*. By definition **origin.x<=corner.x** and **origin.y<=corner.y** define the rectangle. By convention, the right (maximum *x*) and bottom (maximum *y*) edges are excluded from the represented rectangle, so abutting rectangles have no points in common. Thus, *corner* is the coordinates of the first point beyond the rectangle. The data on the display is contained in the **Rectangle** {**0, 0, XMAX, YMAX**} where **XMAX=800** and **YMAX=1024**.

Bitmap

```
typedef struct Bitmap {
   Word      *base;        /* pointer to start of data */
   unsigned  width;        /* width in Words of data area */
   Rectangle rect;         /* rectangle in data area */
   char      *_null;       /* unused, always zero */
} Bitmap;
```

A *Bitmap* holds a rectangular image stored in contiguous memory starting at *base*. Each *width* words of memory form a scan-line of the image. The *rect* argument defines the coordinate system inside the *Bitmap*. Argument *rect.origin* is the location in the *Bitmap* of the upper left-most point in the image and is not necessarily (0,0). Graphical operations performed on a *Bitmap* are clipped to *rect*.

Texture

```
typedef struct Texture {
      Word   bits[32];
} Texture;
```

A *Texture* is a 32×32 dot bit pattern. *Texture*s are aligned to absolute display positions, so adjacent areas colored with the same *Texture* mesh smoothly.

Texture16

```
typedef struct Texture16 {
   short  bits[16];
} Texture16;
```

A *Texture16* is a 16×16 dot bit pattern, similar to a *Texture*. The mouse cursor is a *Texture16*.

Font and *Fontchar*

```
typedef struct Fontchar
{
   short          x;       /* left edge of bits */
   unsigned char  top;     /* first non-zero scan-line */
   unsigned char  bottom;  /* last non-zero scan-line */
   char           left;    /* offset of baseline */
   unsigned char  width;   /* width of baseline */
} Fontchar;

typedef struct Font
{
   short    n;             /* number of chars in font */
   char     height;        /* height of bitmap */
   char     ascent;        /* top of bitmap to baseline */
   long     unused;
   Bitmap   *bits;         /* where characters are stored */
   Fontchar info[1];       /* n+1 character descriptors */
} Font;
```

A *Font* is a character set. The character information is stored in the *Fontchar* structures. The actual images of the characters are stored in the horizontal *Bitmap* bits. A character at point *p* has its upper left-most dot (including empty space above it) in the *Bitmap*, at *p*. Characters in the *Bitmap* abut exactly, so the width of a character *c* is **Font.info[c+1].x-Font.info[c].x**.

NAME

strwidth, jstrwidth – width of character string

SYNOPSIS

#include <font.h>

int strwidth (f, s)

int jstrwidth (s)

Font *f;
char *s;

DESCRIPTION

The *strwidth* function returns the width in full-screen coordinates (pixels) of the null-terminated string *s*, interpreted in the Font **f*. The height of a character string is simply *f->height*.

The call, *jstrwidth(s)*, is equivalent to *strwidth (&defont, s)*.

EXAMPLE

The following code fragment places the width of a string in a large font into a variable called width.

```
#include <dmd.h>
#include <font.h>
    .
    .
    Font      *f;
    int width;

    if( (f = getfont ("/usr/dmd/font/HI.24")) ==  )
        exit();
    width = strwidth (f, "hello world");
    .
    .
    ffree (f);
```

SEE ALSO

getfont(3L), string(3R), stringc(3L), strlen(3R), structures(3R).

NAME

texture – draw Texture in Rectangle in Bitmap

SYNOPSIS

void texture (b, r, t, f)

Bitmap *b;
Rectangle r;
Texture *t;
Code f;

Texture T_grey, T_lightgrey, T_darkgrey;

Texture T_black, T_white, T_background, T_checks;

DESCRIPTION

The *texture* function draws the Texture specified by *t* with function Code *f* in the Rectangle *r* in the Bitmap *b*. The Textures listed above are in ROM and are accessible with the above names.

EXAMPLE

The following routine allows doodling with a Texture.

```
#include <dmd.h>
main()
{
        Rectangle r;
        Point s;
        s.x = 16;
        s.y = 16;
        request (MOUSE);
        for (wait(MOUSE); !button3();
              wait(MOUSE)){
              r.origin = mouse.xy;
              r.corner = add (r.origin, s);
              if( button1() )
                    texture (&display, r,
                          &T_grey, F_STORE);
              if( button2() )
                    texture (&display, r,
                          &T_grey, F_CLR);
        }
}
```

SEE ALSO

jtexture(3R), structures(3R), texture16(3L).

NAME

texture16 – draw Texture16 in Rectangle in Bitmap

SYNOPSIS

void texture16 (b, r, t, f)

Bitmap *b;
Rectangle r;
Texture16 *t;
Code f;

DESCRIPTION

The *texture16* function draws the Texture16 specified by *t* with function Code *f* in the Rectangle *r* in the Bitmap *b*. This function is less efficient than *texture* and should only be used when size is an issue.

EXAMPLE

The following call will fill in the layer with the clock cursor.

```
#include <dmd.h>

        .
        .

texture16 (&display, Drect,
        &C_clock, F_XOR);
```

SEE ALSO

cursor(3R), jtexture(3R), structures(3R), texture(3R).

NAME

transform, rtransform – screen to layer coordinates

SYNOPSIS

Point transform(p)
Point p;

Rectangle rtransform(r)
Rectangle r;

DESCRIPTION

The *transform* function returns the layer coordinates of its argument screen coordinate Point *p*.

The *rtransform* function returns the layer coordinates of its argument screen coordinate Rectangle *r*.

The *transform* and *rtransform* calls are not defined for programs running **stand-alone**.

EXAMPLE

The following code will obtain the screen coordinates of the PtCurrent used by the j-routines.

```
#include <dmd.h>
Point p;
transform(PtCurrent);
```

The following two calls to *segment* and *jsegment* will create the same line.

```
#include <dmd.h>
Point p, q;
    .
   p.x = 100;
   p.y = 100;
   q.x = 300;
   q.y = 300;
   jsegment (p,q,F_XOR);
    .
    .
   segment(&display, transform(p),
           transform(q), F_XOR);
```

SEE ALSO

jsegment(3R), segment(3R).

NAME

cos, sin, atan2 – cosine, sine and arc tangent trigonometric functions

SYNOPSIS

int cos(d)

int sin(d)

int atan2 (x, y)

int d;
int x, y;

DESCRIPTION

The *cos* and *sin* functions return scaled integer approximations to the trigonometric functions. The argument values are in degrees. The return values are scaled so that **cos(0)==1024**.

The *atan2* function returns the approximate arc-tangent of *y/x*. The return value is in integral degrees. The error in approximation may be as large as five degrees.

EXAMPLE

To calculate, for example, the mathematical expression

```
x=x0*cos(d)
```

to calculate a projection, the multiplication must be scaled:

```
x=muldiv(x0, cos(d), 1024)
```

NAME

version – return terminal version number

SYNOPSIS

int version ()

DESCRIPTION

The *version* function returns a hex number which identifies the version of a particular terminal, including the firmware version.

The version number here is the same as the one displayed on the screen during keyboard requested self-test and as a response to the <**ESC**>[**c** escape sequence. There are three fields (f1;f2;f3) defined as follows:

f1	identifies the terminal as a 5620
f2	identifies which firmware and hardware options are installed
f3	identifies firmware release

The integer returned by version is a hex number rather than the above ASCII string for easier parsing by an application program. The hex number has the same three fields in the format 0xf1f2f3, where each field is one byte.

EXAMPLE

For example:

```
version 0x080705
```

is equivalent to version

```
8;7;5
```

which corresponds to DMD Release 2.0.

SEE ALSO

termid(7H).

NAME

proto, pinit, psend, precv – multiplexed channels protocol

SYNOPSIS

```
#include <xtproto.h>

int pinit(channels)
int channels;

void precv(ptr, size)
char *ptr;
int size;

int psend(channel, ptr, size)
int channel;
char *ptr;
int size;

void ptimeout(sig)
int sig;

struct Pchannel pconvs[];

struct Pconfig pconfig;
```

DESCRIPTION

The *xtproto* routines implement a machine-independent driver for full-duplex, multiplexed, multi-buffered channels with guaranteed delivery of ordered data via an 8-bit byte data stream. It is used for communication between the *xt* driver and the DMD terminal when operating under *layers*.

CONFIGURATION

The protocol driver assumes an 8 bit byte data stream described by the field *xfdesc* in the user initialized structure *pconfig*. The user must also supply the addresses of routines with which to write packets to the data stream, and to receive unpacked data from the stream, and control information from the stream. These are initialized in the fields *xfuncp*, *rfuncp* and *rcfuncp*.

The routine **xfuncp* will be invoked with arguments as in the *write*(2) system call. The first argument will be *xfdesc*, followed by a pointer to a packet and its size.

The routines **rfuncp* and **rcfuncp* are invoked with three arguments: the first is a channel number, followed by a pointer to data

from that channel and its size. If **rfuncp* returns a non-zero result, the packet will be negatively acknowledged, causing a later retransmission. The routine **rcfuncp* will only be called with data from a control packet. This data is generated by placing a (non-zero) byte in the *pconfig* field *cdata* during an invocation of the routine **rfuncp*. The control byte will be returned to the remote channel in the acknowledgement packet, there to be delivered by an invocation of **rcfuncp*.

Note: Control data can be delivered asynchronously to stream data, but its delivery is uncertain. It is intended that control data be used primarily as an aid to flow control.

PROTOCOL

The protocol uses packets with a 2-byte header containing sequence number (modulo SEQMOD), channel number, control flag, and data size. The data part of a packet may not be larger than MAXPKTDSIZE. The trailer contains a CRC-16 code in 2 bytes. Each channel is multi-buffered, the number of buffers being defined by the parameter NPCBUFS.

Correctly received packets in sequence are acknowledged with a control packet containing an ACK; however, out of sequence packets generate a control packet containing an NAK, which will cause the retransmission in sequence of all un-acknowledged packets.

Unacknowledged packets are retransmitted after a time-out specified by the *pconfig* field *xtimo* in the same units as the field *tscan* which is the period of a time-out scan. Incomplete receive packets are discarded after a time-out specified by the field *rtimo*. These fields, if zero, will be initialized to a default setting appropriate for a 1200 baud RS-232 data stream. If other data stream speeds are used, the time-out fields should be preset. If *speed* is in bytes per *tscan* unit, then *xtimo* should be set to

```
max (3, (NCHAN*NPCBUFS*sizeof
    (struct Packet)+speed-1)/speed)
```

and *rtimo* to

```
max (2, (sizeof(struct rPacket)
    +speed-1)/speed)
```

INITIALIZATION

Pinit initializes the transmitter and receiver data. Transmitter structures must be declared by the caller in an array

```
struct Pchannel pconvs[NCHAN];
```

with the size NCHAN passed as first argument to *pinit*. NCHAN must not be greater than MAXPCHAN. *Pinit* also enables SIGALRM to call the time-out handler *ptimeout*. *Pinit* returns a non-zero result if an initialization inconsistency is detected.

USAGE

Precv is called with a pointer to data read from the data stream [e.g., via a *read*(2) system call] and its size. It collects a packet or acknowledgement from the incoming data and invokes **rfuncp* (or **rcfuncp*) when a good packet has been identified, calling **xfuncp* to send acknowledgements.

Psend is called with three arguments: the first is a channel number, followed by a pointer to data destined for that channel and its size. *Psend* returns a –1 if the channel is blocked and it cannot accept the data; otherwise, *psend* returns the value returned by **xfuncp*.

In each *Pchannel* structure a field *freepkts* is set to the number of free packet buffers available for the channel. There is also a spare byte, *user*.

DEFAULTS

MAXPCHAN	8 (3 bits in header)
SEQMOD	8 (3 bits in header)
NPCBUFS	2 (maximum SEQMOD/2)
MAXPKTDSIZE	32 (limited by buffering in system)

STATISTICS

Optional statistics are available by recompiling the source with the C compiler flag **–D** *PSTATISTICS*. This declares an array *pstats* of type *struct Pstatistics* of size PS_NSTATS. Additional descriptive strings are initialized by the flag **–D** *PSTATSDESC*. A particular statistic **i** may then be obtained on standard output by:

```
printf ("%s: %ld", pstats[i].descp,
                   (long)pstats[i].count);
```

SEE ALSO

layers(7), xt(4).
alarm(2), read(2), signal(2), write(2) in the *UNIX System V Programmer Reference Manual*.

CONTENTS

NAME

dmda.out – *WE*-32001 object file format

DESCRIPTION

The *dmda.out* file is the output file from the assembler *m32as (1)* and the link editor *m32ld*(1). The resultant file can be executed on the target machine if no errors or unresolved references were found. In no case is the file given UNIX system execute permissions because the object code is for the target machine, not the host machine, on which the file was created.

A *WE*-32001 object file supports user-defined sections and contains extensive information for symbolic software testing. The overall structure of a *WE*-32001 object file is given below. (An optional part of the common object file is not present in *WE*-32001.)

File header.
Section 1 header.
...
Section n header.
Section 1 data.
...
Section n data.
Section 1 relocation.
...
Section n relocation.
Section 1 line numbers.
...
Section n line numbers.
Symbol table.

See *filehdr*(4), *scnhdr*(4), *reloc*(4), *linenum*(4), and *syms*(4) for descriptions of the individual parts. Every section created by *m32as* contains a multiple of four number of bytes; directives to *m32ld* can create a section with an odd number of bytes.

A set of functions exist to manipulate *WE*-32001 object files. See *ldfcn*(4) and its associated references for more information.

SEE ALSO

dmdcc(1), m32as(1), m32ld(1).
ldfcn(4), filehdr(4), linenum(4), reloc(4), scnhdr(4), syms(4) in the *UNIX System V Programmer Reference Manual*.

NAME

dmdar – common archive file format

DESCRIPTION

The archive command *dmdar*(1) is used to combine several files into one. Archives are used mainly as libraries to be searched by the link editor *32ld*(1).

Each archive begins with the archive magic string.

```
#define  ARMAG   " !<arch>\n"        /* magic string */
#define  SARMAG  8                   /* length of magic string */
```

Each archive which contains common object files (see *dmda.out*(4)) includes an archive symbol table. This symbol table is used by the link editor *m32ld*(1) to determine which archive members must be loaded during the link edit process. The archive symbol table (if it exists) is always the first file in the archive (but is never listed) and is automatically created and/or updated by *dmdar*.

Following the archive magic string are the archive file members. Each file member is preceded by a file member header which is of the following format:

```
#define  ARFMAG   " `\n"        /* header trailer string */

struct  ar_hdr                  /* file member header */
{
   char    ar_name[16];         /* '/' terminated file member name */
   char    ar_date[12];         /* file member date */
   char    ar_uid[6];           /* file member user identification */
   char    ar_gid[6];           /* file member group identification */
   char    ar_mode[8];          /* file member mode (octal) */
   char    ar_size[10];         /* file member size */
   char    ar_fmag[2];          /* header trailer string */
};
```

All information in the file member headers is in printable ASCII. The numeric information contained in the headers is stored as

decimal numbers (except for *ar_mode* which is in octal). Thus, if the archive contains printable files, the archive itself is printable.

The *ar_name* field is blank-padded and slash (/) terminated. The *ar_date* field is the modification date of the file at the time of its insertion into the archive. Common format archives can be moved from system to system as long as the portable archive command *dmdar*(1) is used.

Each archive file member begins on an even byte boundary; a newline is inserted between files if necessary. Nevertheless the size given reflects the actual size of the file exclusive of padding.

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

If the archive symbol table exists, the first file in the archive has a zero length name (i.e., **ar_name[0]** == '/'). The contents of this file are as follows:

- The number of symbols. Length: 4 bytes.
- The array of offsets into the archive file. Length: 4 bytes * "the number of symbols".
- The name string table. Length: *ar_size* – (4 bytes * ("the number of symbols" + 1)).

The number of symbols and the array of offsets are managed with *sgetl* and *sputl*. The string table contains exactly as many null terminated strings as there are elements in the offsets array. Each offset from the array is associated with the corresponding name from the string table (in order). The names in the string table are all the defined global symbols found in the common object files in the archive. Each offset is the location of the archive header for the associated symbol.

SEE ALSO

dmdar(1), m32ld(1), m32strip(1), sputl(3X), dmda.out(4).

BUGS

The *m32strip*(1) command will remove all archive symbol entries from the header. The archive symbol entries must be restored via the **ts** option of the *dmdar*(1) command before the archive can be used with the link editor *m32ld*(1).

NAME

plot – graphics interface

DESCRIPTION

Files of this format are produced by routines described in *plot*(3X) and are interpreted for various devices by commands described in *tplot*(1G). A graphics file is a stream of plotting instructions. Each instruction consists of an ASCII letter usually followed by bytes of binary information. The instructions are executed in order. A point is designated by four bytes representing the *x* and *y* values; each value is a signed integer. The last designated point in an **l**, **m**, **n**, or **p** instruction becomes the "current point" for the next instruction.

Each of the following descriptions begins with the name of the corresponding routine in *plot*(3X).

m Move: The next four bytes give a new current point.

n Cont: Draw a line from the current point to the point given by the next four bytes. See *tplot*(1G).

p Point: Plot the point given by the next four bytes.

l Line: Draw a line from the point given by the next four bytes to the point given by the following four bytes.

t Label: Place the following ASCII string so that its first character falls on the current point. The string is terminated by a new-line.

e Erase: Start another frame of output.

f Linemod: Take the following string, up to a new-line, as the style for drawing further lines. The styles are "dotted", "solid", "longdashed", "shortdashed", and "dot-dashed". This is effective only for the **–T4014**, **–Tver**, and **–T5620** options of *tplot*(1G) (Tektronix 4014 terminal, Versatec plotter, and DMD 5620 terminal).

s Space: The next four bytes give the lower left corner of the plotting area; the following four give the upper right corner. The plot will be magnified or reduced to fit the device as closely as possible.

Space settings that exactly fill the plotting area with unity scaling appear below for devices supported by the filters of *tplot*(1G). The upper limit is just outside the plotting area. In every case the plotting area is taken to be square; points outside may be displayable on devices whose face is not square.

DASI 300	space(0, 0, 4096, 4096);
DASI 300s	space(0, 0, 4096, 4096);
DASI 450	space(0, 0, 4096, 4096);
Tektronix 4014	space(0, 0, 3120, 3120);
Versatec plotter	space(0, 0, 2048, 2048);
DMD 5620	space(0, 0, 800, 800);

SEE ALSO

plot(3X), tplot(1G).
graph(1G) in the *UNIX System V User Reference Manual.*
gps(4), term(5) in the *UNIX System V User Reference Manual.*

NAME

xt – Multiplexed tty driver for DMD layers

DESCRIPTION

The *xt* driver provides virtual *tty* circuits multiplexed onto real *tty* lines. It interposes its own channel multiplexing protocol as a line discipline between the real device driver and the standard *tty* line disciplines.

Virtual *tty* circuits are named by inodes in the directory **/dev/xt**. Filenames are three digits, where the first two represent the *channel group* and the last represents the virtual tty number (0-7) of the channel group. Allocation of a new channel group is done dynamically by attempting to open a name ending in '*0*' with the O_EXCL flag set. After a successful open, the *tty* file onto which the channels are to be multiplexed should be passed to *xt* via the XTIOCLINK *ioctl*(2) command. Afterwards, all the channels in the group will behave as normal *tty* files, with data passed in packets via the real *tty* line.

The *xt* driver implements the protocol described in *xtproto*(3C) and in *layers*(7). Packets are formatted as described in *xtproto*(3C), while the contents of packets conform to the description in *layers*(7).

There are three groups of *ioctl*(2) commands implemented by *xt*. The first group contains all the normal *tty ioctl* commands described in *tty*(7), with the addition of the following:

TIOCEXCL Set *exclusive use* mode: No further opens are permitted until the file has been closed.

TIOCNXCL Reset *exclusive use* mode: Further opens are once again permitted.

The second group concerns control of the DMD layers and is described in the header file **/usr/include/sys/jioctl.h**.

The commands are as follows:

JTYPE, JMPX Both return the value JMPX. This is used to identify a DMD.

JBOOT, JTERM Both generate an appropriate command packet to the DMD on channel 0. May return the error

code EIO if the system *clist* is empty.

JTIMO, JTIMOM — JTIMO specifies the timeouts in seconds, and JTIMOM in milliseconds. This is invalid except on channel 0. May return the error code EIO if the system *clist* is empty.

JWINSIZE — Requires an argument being the address of a *jwinsize* structure. The current layer window sizes are copied to the structure.

The third group of commands concerns the configuration of *xt*, and is described in the header file **/usr/include/sys/xt.h**.

The commands are as follows:

XTIOCTYPE — Returns the value *XTIOCTYPE*.

XTIOCLINK — Requires an argument being a structure *xtioclm* describing the file to be multiplexed, and the maximum number of channels allowed. Invalid except on channel 0. This command may return one of the following errors:

- EINVAL — *nchans* has an illegal value.
- ENOTTY — *fd* does not describe a real *tty* device.
- ENXIO — *linesw* is not configured with *xt*.
- EBUSY — An XTIOCLINK command has already been issued for the channel group.
- ENOMEM — There is no system memory available for allocating to the *tty* structures.
- EIO — The JTIMOM packet described above could not be delivered.

XTIOCTRACE — Requires an argument being the address of a *Tbuf* structure. The structure is filled with the contents of the driver trace buffer. Tracing is enabled. This command is invalid if tracing is not configured.

XTIOCNOTRACE — Tracing is disabled. This command is invalid if tracing is not configured.

XTIOCSTATS Requires an argument being the address of an array of size *S_NSTATS*, of type *Stats_t*. The array is filled with the contents of the driver statistics array. This command is invalid if statistics are not configured.

XTIOCDATA Requires an argument being the address of a maximum sized *Link* structure. The structure is filled with the contents of the driver *Link* data. This command is invalid if not configured.

FILES

/dev/xt/??[0-7]	Multiplexed special files.
/usr/include/sys/jioctl.h	Packet command types.
/usr/include/sys/xtproto.h	Channel multiplexing protocol definitions.
/usr/include/sys/xt.h	Driver specific definitions.

SEE ALSO

xtproto(3C), layers(7).
ioctl(2), open(2) in the *UNIX System V User Reference Manual.*
tty(7) in the *UNIX System V Administrator Reference Manual.*

CONTENTS

NAME

jagent – DMD agent control device

SYNOPSIS

#include <jioctl.h>

ioctl (cntlfd, JAGENT, &arg)

int cntlfd;
int JAGENT;
struct bagent arg;

DESCRIPTION

The *ioctl* utility with *JAGENT* allows a host program to send information to the DMD terminal. *Ioctl* has three arguments:

i) *cntlfd*: the DMD control channel file descriptor

ii) *JAGENT*: a defined request to call the *agent* routine

iii) *arg*: the address of the *bagent* structure.

The call uses the following structure, defined in <**jioctl.h**>:

```
struct bagent {
  int   size;  /* size of src in & dest out */
  char  *src;  /* the source byte string */
  char  *dest; /* the destination byte string */
};
```

The *src* pointer points to a byte string which is sent to the DMD. When *ioctl* returns, *dest* points to a byte string which was returned by the DMD agent. The *size* argument is the length of the *src* string when *ioctl* is called. Upon return, *size* is the length of the destination byte string.

RETURN VALUE

If an error has occurred during the call to *ioctl*, a value of –1 is returned. Upon successful completion of *ioctl*, the size of the destination byte string is returned.

SEE ALSO

agent(3R), hagent(3H).

NAME

layers – Protocols used between the UNIX system and DMD layers

SYNOPSIS

#include <sys/jioctl.h>

DESCRIPTION

The *layers* utilities are asynchronous windows supported by an operating system in the DMD terminal. Communication between the UNIX system processes and DMD processes running in layers occurs via multiplexed channels managed by the respective operating systems using a protocol as specified in *xtproto*(3C).

The contents of packets transferring data between a UNIX system process and a *layer* are asymmetric. Data sent from the UNIX system to a particular layer process is undifferentiated, and it is up to the *layer* process to interpret the contents of packets.

Control information for *layer* processes is sent via channel 0, and process 0 in the DMD performs the designated functions on behalf of the process connected to the designated channel. These packets take the form:

```
command, channel
```

except for *timeout* and *jagent* information which take the form

```
command, data...
```

The commands are the bottom eight bits extracted from the following I/O control codes:

JBOOT Prepare to load a new DMD program into the designated *layer*.

JTERM Kill the *layer* program, and restore the window program.

JTIMO Set the timeout parameters for the protocol. The data consists of two bytes, being the value of the receive timeout in seconds, followed by the value of the transmit timeout in seconds.

JTIMOM Set the timeout parameters for the protocol. The data consists of four bytes in two groups. The first group is the value of the receive timeout in milliseconds, the low eight bits followed by the high eight bits. The second group is the value of the transmit timeout.

JAGENT Send a source byte string to the DMD agent routine and wait for a destination byte string to be returned. The data is in the following form.

```
struct bagent {
 int  size;    /* size of src string in & dest out */
 char * src;   /* address of source byte string */
 char * dest;  /* address of destination byte string */
};
```

Packets from *layers* to the UNIX system all take the following form:

```
command, data...
```

The commands are as follows:

C_SENDCHAR Send the next byte to the UNIX system process.

C_NEW Create a new UNIX system process group for this *layer*. Remember the window size parameters for this *layer*. The data for this command is in the form described by the *jwinsize* structure.

C_UNBLK Unblock transmission to this *layer*. There is no data associated with this command.

C_DELETE Delete a UNIX system process group attached to this *layer*. There is no data associated with this command.

C_EXIT Exit. Kill all UNIX system process groups and terminate the session. There is no data

associated with this command.

C_BRAINDEATH Layer program has died; send a terminate signal to the UNIX system process group. There is no data associated with this command.

C_SENDNCHARS The remaining data are characters to be passed to the UNIX system process.

C_RESHAPE The *layer* has been reshaped. Change the window size parameters for this *layer*. The data takes the same form as for the *C_NEW* command.

SEE ALSO

layers(1), xt(4), xtproto(3C).

NAME

SENDTERMID – query the terminal ID number from the host

SYNOPSIS

#define SENDTERMID "\033[c"

#define TERMIDSIZE 9

DESCRIPTION

Host programs needing to know what version of ROM is in the terminal should include the code in the example below. For DMD 2.0 terminals, the string "8;7;5" will be returned.

EXAMPLE

The following routine will get the terminal ROM ID when passed the address of a buffer to place the ROM ID string.

```
#define SENDTERMID "\033[c"
#define TERMIDSIZE 9

getromid(buf)
char  *buf;
{
      int cnt, nr;

      write (1, SENDTERMID, 3);
      cnt = 0;
      while (cnt < TERMIDSIZE){
            nr = read (0, buf+cnt,
                  TERMIDSIZE-cnt);
            if (nr > 0)
                  cnt += nr;
      }
}
```

WARNINGS

This escape sequence will only work with the default terminal emulator under *layers*(1). Downloaded programs (e.g., *hp2621(1)*) will not recognize it.

NAME

termio – how to setup termio structure for host portions of DMD applications

SYNOPSIS

#include <sys/termio.h>

struct termio tty;

DESCRIPTION

DMD applications with host portions needing to pass binary data between the DMD and the host will need to include the following code in their host source.

EXAMPLE

The *ttysave* function will save the terminal characteristics and then change them to permit the passing of binary data. The *ttyrestore* function will restore the terminal characteristics and should be called when the host program has completed.

```
#include <stdio.h>
#include <sys/termio.h>

struct termio stty, wtty;

ttysave()
{
        if ( ioctl(1, TCGETA, &stty) == -1 ){
                fprintf (stderr, "ioctl received
                        error on TCGETA\n");
                exit ();
        }
        wtty.c_iflag = IGNBRK;
        wtty.c_cflag = (stty.c_cflag&
                (CBAUD|CLOCAL)) |CS8 |CREAD;
        wtty.c_cc[VMIN] = 1;
        if ( ioctl (1, TCSETA, &wtty) == -1 ){
                fprintf (stderr, "ioctl received
                        error on TCSETA\n");
                exit ();
        }
}
```

```
ttyrestore()
{
        if ( ioctl (1, TCSETA, &stty) == -1 ){
                fprintf (stderr, "ioctl received
                        error on TCSETA\n");
                exit ();
        }
}
```

SEE ALSO

ioctl(2).

Index

A

B

C

D

E

F

G

H

I

J

K

L

M

P

Q

R

S

T

V

W

X

Your comments and suggestions are appreciated and will help us to provide the best documentation for your use.

1. How would you rate this document for COMPLETENESS? (Please Circle)

   Excellent Adequate Poor
   4 ----------3 ----------2 ----------1 ----------0

2. Identify any information that you feel should be included or removed.

   _____

   _____

3. How would you rate this document for ACCURACY of information? (Please Circle)

   Excellent Adequate Poor
   4 ----------3 ----------2 ----------1 ----------0

4. Specify page and nature of any error(s) found in this document.

   _____

   _____

5. How would you rate this document for ORGANIZATION of information? (Please Circle)

   Excellent Adequate Poor
   4 ----------3 ----------2 ----------1 ----------0

6. Describe any format or packaging problems you have experienced with this document.

   _____

   _____

7. Do you have any general comments or suggestions regarding this document?

   _____

   _____

8. We would like to know a little about your background as a user of this document:

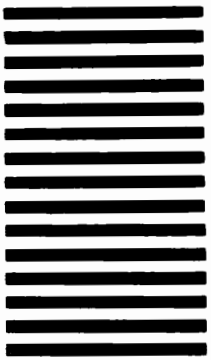
   A. Your job function _____

   B. Number of years experience with computer hardware: operation _____ , maintenance _____ .

   C. Number of years experience with computer software: user _____ , programmer _____ .

Your Name _____ Phone No. _____
Company _____
Address _____
City & State _____ Zip Code _____

NO POSTAGE
NECESSARY
IF MAILED
IN THE
UNITED STATES



BUSINESS REPLY MAIL
FIRST CLASS PERMIT NO 1999 GREENSBORO N C

POSTAGE WILL BE PAID BY ADDRESSEE

DOCUMENTATION SERVICES
2400 Reynolda Road
Winston-Salem, N.C. 27106-9989

Do Not Tear—Fold Here and Tape

Your comments and suggestions are appreciated and will help us to provide the best documentation for your use.

1. How would you rate this document for COMPLETENESS? (Please Circle)

Excellent		Adequate		Poor
4	3	2	1	0

2. Identify any information that you feel should be included or removed.

_____

_____

3. How would you rate this document for ACCURACY of information? (Please Circle)

Excellent		Adequate		Poor
4	3	2	1	0

4. Specify page and nature of any error(s) found in this document.

_____

_____

5. How would you rate this document for ORGANIZATION of information? (Please Circle)

Excellent		Adequate		Poor
4	3	2	1	0

6. Describe any format or packaging problems you have experienced with this document.

_____

_____

7. Do you have any general comments or suggestions regarding this document?

_____

_____

8. We would like to know a little about your background as a user of this document:

A. Your job function _____

B. Number of years experience with computer hardware: operation _____ , maintenance _____ .

C. Number of years experience with computer software: user _____ , programmer _____ .

Your Name _____ Phone No. _____
Company _____
Address _____
City & State _____ Zip Code _____

NO POSTAGE
NECESSARY
IF MAILED
IN THE
UNITED STATES

BUSINESS REPLY MAIL
FIRST CLASS PERMIT NO 1999 GREENSBORO N C

POSTAGE WILL BE PAID BY ADDRESSEE

DOCUMENTATION SERVICES
2400 Reynolda Road
Winston-Salem, N.C. 27106-9989

Do Not Tear—Fold Here and Tape