UNITED STATES DEPARTMENT OF THE INTERIOR

GEOLOGICAL SURVEY

Technical Manual for a UNIX-based Device-Independent Vector Graphic System

by

Gerald I. Evenden[1]

Open-File Report 91–2

This report is preliminary and has not been reviewed for conformity with U.S. Geological Survey editorial standards. Use of tradenames is for purposes of identification only and does not constitute endorsement by the U.S. Geological Survey.

January 2, 1991

[1] Woods Hole, MA

Contents

Abstract 1

Introduction 1

Source Distribution 1

Creating metagraphic data 1

System Coordinates 2

Plot initialization 2

Pen Initialization and Control 3

Select pen 4

Pen Positioning 4

Character String Plotting 5

Symbol Plotting 7

Line Plotting 7

Symbol Line Plotting 8

Request to Plotter 9

Metagraphic file format 10

Plotter drivers 10

Internal drivers 10

External drivers 13

Installing device drivers 15

Fonts 16

Hershey Drafting System 16

Font structure 17

Typesetting fonts 17

References 18

Appendix A—Standard Fonts 19

Appendix B–UNIX Manual Style Documentation 35

Figures

1 Example of basic vector plotting calls 5

2 Example of basic string plotting calls 7

3 Examples of extended line plotting calls 9

4 Stroke pattern for the Hershey Simplex Roman letter A, character 501. 16

5 Stroke pattern for the Hershey Triplex Roman letter A, character 3001. 17

Tables

1 Metagraphic command byte. 10

2 External driver communication stream. 14

3 Contents of **symgen** selection file `sr.D` for making **plotter** font file `-sr`. 19

4 Symbols for font `-3osw` 20

5 Symbols for Cartographic font (`-cartr`). 20

6 Symbols for Cyrillic font (`-cc`). 21

7 Symbols for font (`-cgi`). 21

No page ii

8 Symbols for font **-cip** 22
9 Symbols for font **-cri** 22
10 Symbols for font **-crp** 23
11 Symbols for font **-cscp** 23
12 Symbols for font **-din** 24
13 Symbols for font **-dr** 24
14 Symbols for font **-engl** 25
15 Symbols for font **-germ** 25
16 Symbols for font **-ital** 26
17 Symbols for font **-ksym1** 26
18 Symbols for font **-ksym2** 27
19 Symbols for font **-osw** 27
20 Symbols for font **-sg** 28
21 Symbols for font **-sr** 28
22 Symbols for font **-sscp** 29
23 Symbols for font **-sym1** 29
24 Symbols for font **-tr** 30
25 Symbols for font **-tri** 30
26 Symbols for typesetting font **-nrR** 31
27 Symbols for typesetting font **-nrB** 31
28 Symbols for typesetting font **-nrI** 32
29 Symbols for typesetting font **-nrBI** 32
30 Symbols for typesetting font **-nrS** 33

Technical Manual for a UNIX-based Device-Independent Vector Graphic System

Gerald I. Evenden

Abstract

Technical details of a basic device independent vector graphic system for the UNIX operating system environment are described. The system uses a metagraphic data stream created by applications programs employing documented library procedures. This stream can be either saved as a file, transported to other computer systems or immediately interpreted and displayed by program **plotter** which converts the data to specific graphic commands required by the selected plotter. Several hard-copy plotters, interactive graphic terminals as well as X11 Windows are currently supported and emphasis is placed upon detailed description of how additional graphic devices can be added to the system. Description of the metagraphic stream and how additional character fonts can be created are also given as well as tabulations of current standard character fonts.

Introduction

This is a technical description of a vector graphic system for UNIX application programs that is designed to provide readily adaptable software for a wide variety of display hardware and thus enhance the transportability of the application software using it. The system consists of a library of C functions to be executed by the applications program that create a device independent metagraphic data stream to control a generic vector plotter. This data stream can be either directly piped to the program **plotter** which converts the information to the commands to operate a specific plotting device or saved in a file (usually termed a "deferred" or "overlay" file) for later processing by **plotter**.

To achieve flexibility, only the most primitive graphic device is required for making vector plots: a pen capable of being moved in a "pen up" or nondrafting mode and moved in a "pen down" or line drawing mode. Although most plotting hardware provide more facilities which may be accessed by mechanisms within the system, any metagraphic file can be displayed on any device—suffering only the lack of enhancements of more sophisticated displays. Character drafting is performed by the system as well as line smoothing, dashing and various other features. For interactive graphic devices a mechanism for obtaining cursor coordinates from the screen is also provided.

The purpose of this report is to document the C procedures used by application programs, aspects of various system features and a detailed description of how additional plotting devices can be added to the system. Details and display of the standard character sets distributed with the system are tabulated along with instructions as to how new ones can be created.

This graphic system is designed to provide public-domain vector graphic software in support of the MAPGEN-PLOTGEN system (Evenden and Botbol, 1985) so that it can be transported without reliance upon graphic packages that may not be available at new host sites. Although this system does not represent state of the art graphic techniques and lacks some features of more modern graphic packages, it has shown a robustness and flexibility that has caused it to retain its usefulness.

Source Distribution

All software discussed in this report may be obtained from the author. This consists of a file subsystem containing appropriate supplementary "README" and installation procedures for creating program **plotter**, the application library *libgraph.a*, and the standard fonts used by **plotter**. In addition, *graphics.h* header file, standard UNIX style manual source files, test programs and expanded Hershey character definition tables are included.

Creating metagraphic data

This section is concerned with the programming aspects of creating a metagraphic stream that will be either piped directly to program **plotter**, or saved in overlay files for later display. There are eight entries in the library file *libgraph.a* for execution by application programs to generate a metagraphic stream and to control the execution of program **plotter**:

```
#include <graphics.h>
int plotopen((char **)argl)
void plotend()
int defopen([(char *)type,] (char *)name)
void defclose()
void pltflush()
plotopt((int)opt [, arg])
void pxyxmit((int)opt, (long)x, (long)y)
ANSWR *plotreq((int)opt)
```

Usage of argument opt is usually by means of acronymic **#defines** in the file *graphics.h*. The argument **arg** may not be present for some **plotopt** calls and, when used, its type is dependent upon **arg**: **int**, **long** or **char ***. Because *graphics.h* will probably not be in the standard UNIX system */usr/include* directory, the **-I***path* option pointing to an appropriate directory will be required when compiling application programs. Procedures **plotopt**, **pxyxmit** and **plotreq** are the only entries that generate metagraphic data.

System Coordinates

The system's internal coordinates compose a non-negative, integral cartesian x–y system with a range of 0–8,388,607 counts in both axes and coordinates 0–0 are always at the lower left-hand corner of the plot. Relationship between these units and the units used by specific plotting devices are dependent upon scaling performed by **plotter**'s device output procedures and run-time switches. For consistency of application software scaling, all hard copy devices distributed with the system are considered to have a precision of 200 counts/cm (50μ resolution) irrespective of their true capabilities—sufficient for most publication applications.

For terminal screen devices, the internal coordinates are converted directly to the pixel range of the device (typically in the range of 1,000 counts). A different scale factor for one axis is required for displays that do not have a 1:1 aspect ratio. Appropriate scaling of the metagraphic data on terminal devices can be performed by inquiries to program **plotter** to determine the screen size and sending rescaling information prior to processing the graphic stream.

Because this system is primarily designed for hardcopy vector output, writers of application software should scale data to the internal coordinate system based upon the hardcopy resolution and size of the desired plot. If plots are to be viewed on an interactive device they can be readily scaled down to fit the maximum range of the device.

Plot initialization

The following entries to the graphics system pertain to initialization and non-graphic aspects of the system. When creating a plot either a direct link with the program **plotter** can be established by executing **plotopen** and/or the metagraphics can be output to a deferred plot file by executing **defopen**. If neither opening procedure is called, subsequent calls to the other procedures are NO-OP's.

Direct link to program plotter

A direct link to program **plotter** is open and closed by the respective procedures **plotopen** and **plotend**. **Plotopen**'s argument **argl** is an array of character string pointers passing information to program **plotter**'s **argv** main entry. The size of **argl** must be at least three elements and in all cases the first two will be ignored (they are modified by **plotopen**) and the last entry **must** be null to signify the end of the parameter list. Entries in this list may be any of the options and parameters recognized by program **plotter**, including overlay file names. **Plotopen** will return a 0 for a successful operation, otherwise the linkage failed.

If **plotter** is to accept data from the parent program or be used interactively then the pair of arguments **"-i"** and **"."** must appear sequentially in the argument list. Otherwise, **plotter** expects input only from specified overlay files and **plotopen** will return after completion of the spawned process.

Typical usage of these procedures is summarized as:

```
static char *argl[MAXARG] ={0,0,"-i",".",0};
   ...
   /* open link with plotter */
   if (plotopen(argl)) {
      fprintf(stderr,"plotter link failure\n");
      ...
   }
   /* successful, do graphics */
   ...
   plotend(); /* finished with plotter */
   ...
```

where the user-defined value of **MAXARG** is chosen large enough to provide space in **argl** for supplementary arguments.

If the plotting device name option (**-d**) is omitted, the graphics system assumes that the current controlling teletype is a graphics device and selects the device driver based on the environment setting of **GTERM** or, if not given, **TERM**. Program **plotter** inherits the same status of *stdout* as the parent program so that if another output file is to be used by **plotter** then the **-o** option in **argl** must be used.

Deferred plot file control

When a deferred plot file is opened by a call to **defopen** all metagraphics generated by subsequent calls to **plotopt** or **pxyxmit** are output to this file until it is closed by **defclose**. As with **plotopen**, a 0 value will be returned if the deferred file was successfully opened. Both **plotopen** and **defopen** may be used concurrently to allow monitoring of the creation of a deferred file on a terminal.

An optional string argument begining with a - may precede the file name argument where the characters following the - are the **fopen**(3) *type* argument. For example:

```
defopen("-a", "myfile");
```

will append metagraphic data to **myfile** if it exists or create a new file. When the argument is omitted a **w** type is assumed which will create or overwrite an existing file. The hyphen is used as an indicator that this is the *type* argument and not the file name.

Flush current contents of buffers

```
(void)pltflush();
```

This call is occasionally required to ensure that the current contents of the buffers have been sent to **plotter** and/or the overlay file.

Include deferred file

```
(void)plotopt(INCL, (char *)name);
```

A previously created deferred plot file is be included in the current plot.

Clear graphics area

```
(void)plotopt(ERASE);
```

This device dependent option will erase the screen of interactive graphics devices. It is ignored when applied to hardcopy devices.

Disable graphics mode

```
(void)plotopt(DISABLE);
```

Many screen terminals are capable of selectable and independent text and graphic modes. This operation returns the terminal to the text mode for interactive text-keyboard process control. These terminals will automatically switch to the graphic mode on the receipt of the next graphic command. It is ignored by hardcopy devices. Note: this call should be followed by a **pltflush** call.

Reset plot scaling

```
(void)plotopt(RESCALE, (char *)value);
```

If no pens are active the basic scaling of the device (see **plotopen**) may be changed to the positive value of the decimal number expressed as a string in **value**. It is necessary to format the value in ASCII since the metagraphic stream has no provisions for fractional numbers. Typically this operation is performed immediately after **plotopen** and a **plotreq**(**P_SIZE**) so that custom scaling can be automatically performed by the application program.

Reset x–y base register

```
(void)plotopt(CBASE);
```

Execution of this option will clear **plotter's** x–y registers used to reconstitute the differential x–y data in the metagraphic stream. Usage of the option applies to cases where the metagraphic files may be concatenated by non-graphics software (i.e. a *cat*(1) command) and precedes any calls which transmit x–y data.

Offset registers.

```
(void)plotopt(BASEX, (long)value);
(void)plotopt(BASEY, (long)value);
```

These offset registers will shift all x–y non-relative coordinate data from the normal 0,0 origin. The user may alter the axis offsets with the **BASEX** and **BASEY** values which are algebraically added to the respective coordinate data. Normal application of this option(s) is for zooming operations on metagraphic files.

Special driver control

```
(void)plotopt(SPECIAL, (char *)str);
```

The contents of the null terminated character string **str** are passed directly to the device driver procedure without interpretation by program **plotter**. Contents and meaning of **str** are determined by device driver documentation. Use of this call will limit the device independence of a metagraphic stream but it allows a great deal of flexibility for applications needing access to special plotter features.

Pen Initialization and Control

Before any plotting operations, line drafting, string or symbol posting can be done, a **pen** must be defined. The unqualified term pen is defined in this report as a logical entity and not in terms of the true, physical pen on the plotting device. Many pens can be specified, each with its own attributes which can operate and perform plotting operations totally independent of the other pens. The following set of operations provide for basic pen initialization and setting of non-graphic attributes.

Initialize pen

```
(void)plotopt(NEWPEN, (char*)name);
```

Each pen is given a user defined name of up to 31 characters (more may be used, but the high order characters are ignored). If desired, an existing pen may be used as a template to initialize automatically various options by following the new pen's name in the string with a : and the name of an existing pen's attributes to be copied to the new pen's attibutes. For example:

```
(void)plotopt(NEWPEN, (char*)"penB:penA");
```

will initialize **penB** with the current attributes of **penA**.

Select pen

```
(void)plotopt(SPEN, (char *)name);
```

Any existing pen may be selected by this option. All subsequent graphics operations now apply to this pen.

Delete current pen

```
(void)plotopt(DELPEN);
```

This option removes the currently selected pen. If there are other active pens then the pen selected before the current, deleted pen becomes the current pen, but the user is advised to issue a **SPEN** after this operation.

Link x–y

```
(void)plotopt(LINKXY, (char*)name);
```

The x–y coordinates of the current pen may be linked to those of another pen selected by **name**. Several pens may be linked in this manner. This feature provides for a variety of line and character attributes assigned to different pens tracking the same set of x–y coordinates.

Unlink x–y of current pen

```
(void)plotopt(DELINK);
```

This option is the inverse of **LINKXY** so that the coordinates of the current pen are controlled independent of any other pen.

Window range

```
(void)plotopt(WXL, (long)xlow);
(void)plotopt(WXH, (long)xhi);
(void)plotopt(WYL, (long)ylow);
(void)plotopt(WYH, (long)yhi);
```

Xlow, **xhi**, **ylow** and **yhi** set the respective boundaries of the window for all graphics operations of the current pen. The **low** value must be less than the respective axis **hi** value. Values less than 0 or larger than the size of the plot device are respectively converted to 0 or the size of the device. When a pen is initialized without the attribute copy option the window boundaries are set to the limits of the device.

Mechanical pen selection

```
(void)plotopt(MPEN, (long)mpen);
```

This option is dependent upon the plotting device. For most devices this option will select different mechanical pens which may be of varying color and/or line width. **Mpen** values less than 256 may be mapped to new values by **plotter** runline options.

Pen Positioning

All character, symbol and line drafting is based upon the x–y positioning of the current pen. This positioning may be done in two ways: as an absolute position or as a position relative to the last absolute coordinate value. In either case, the pen may be moved to the new coordinates with or without drafting a line.

Plotter pen motion

```
(void)pxyxmit(_PENUP, (long)x, (long)y);
(void)pxyxmit(_PENUP+_REL, (long)x, (long)y);
(void)pxyxmit(0, (long)x, (long)y);
(void)pxyxmit(_REL, (long)x, (long)y);
```

The above entries move the pen to the specified coordinates. If **_REL** is specified, the **x** and **y** coordinates are relative to the previous pen position. When **_PENUP** is specified the pen motion does not cause a line to be drafted (i.e. a "dark vector"), otherwise the characteristics of the line drafted are defined by the factors set in the line plotting section. The macros **moveto**, **relmoveto**, **lineto** and **rellineto** in the *graphics.h* file may be used in lieu of the above respective **pxyxmit** calls.

Simple line drafting example

The following listing is an example program showing basic program initialization and both relative and absolute vector drafting. For creating a display appropriate for this publication the size of the plot will be limited to the column width of 20 picas (8.47cm) and about 2.5″ (6.35cm) high. Since **plotter**'s PostScript driver expects 200 count/cm coordinates the respective maximum x–y data values should be about 1692 and 1280.

```
#include <graphics.h>
#define XMAX 1692
#define YMAX 1280
#define CM 200
#define T 999999
typedef struct { long x, y; } XY;
/* some objects */
   static XY
neatline[] = { 0,0, XMAX,0, 0,YMAX, -XMAX,0,
   0,-YMAX, 0,0, T,0 },
box[]={-1,-1,2,0,0,2,-2,0,0,-2,1,1,T,0},
triangle[]={-1,-1,2,0,-1,2,-1,-2,1,1,T,0},
diamond[]={0,-2,1,2,-1,2,-1,-2,1,-2,0,2,T,0};
   static void /* plot objects */
do_obj(f, size) XY *f; double size; {
   relmoveto(f->x * size, f->y * size);
   for ( ++f ; f[1].x != T ; ++f)
      rellineto(f->x * size, f->y * size);
   relmoveto(f->x * size, f->y * size);
}
main(argc, argv) char **argv; {
     /* open plot and exit on failure */
   if (defopen(argv[1])) perror(argv[1]), exit(1);
   plotopt(CBASE);       /* force base and      */
   plotopt(ERASE);       /* clear screen        */
   plotopt(NEWPEN,"A");  /* select pen          */
   plotopt(WXH, XMAX);   /* and limit vector    */
   plotopt(WYH, YMAX);   /* range               */
   moveto(0, 0);         /* plot a neatline     */
   do_obj(neatline, 1.);
   moveto(310, 310);     /* plot some nested    */
   do_obj(box, 300.);    /* boxes               */
   do_obj(box, 250.);
   do_obj(box, 200.);
   moveto(XMAX/2, YMAX/2);  /* move around and  */
   do_obj(box, 50.);        /* plot more boxes  */
   lineto(XMAX*.75, YMAX*.75); /* join with a   */
   do_obj(box, 100.);       /* line             */
   moveto(100, YMAX*.75);   /* plot some other  */
   do_obj(diamond, 100.);   /* shapes           */
   moveto(800, YMAX*.5);
   do_obj(triangle, 150.);
   moveto(800, YMAX*.9);    /* this one will be */
   do_obj(diamond, 100.);   /* clipped          */
   moveto(.2*XMAX, .2*YMAX); /* simple line     */
   lineto(.8*XMAX, .5*YMAX);
   defclose(); /* done with plot */
}
```

The overlay created by the above program is displayed in figure 1.

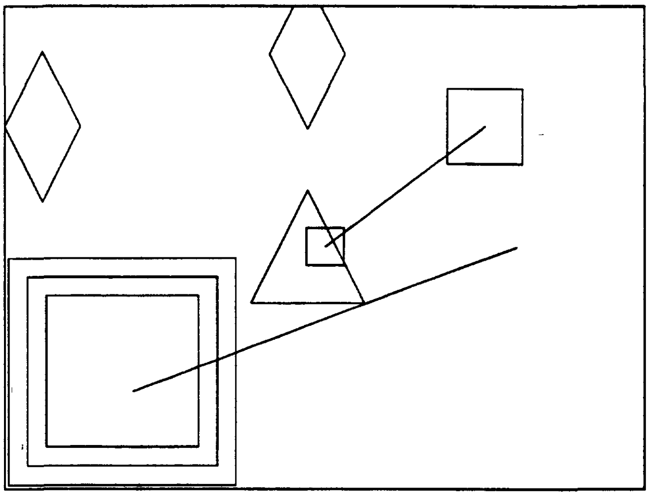

Figure 1: Example of basic vector plotting calls

Character String Plotting

This graphics system provides a comprehensive set of character fonts and control to facilitate the generation of graphics text. However, the programmer must perform at least two steps in anticipation of text plotting: select a font and establish scaling. Remember that all subsequently described attributes are associated with only the currently selected pen.

A second factor associated with text plotting is the **net** coordinate position where the string will be located. The **pxyxmit** command will determine the base position for the text data but the **net** position is a function of **XOFF**, **YOFF** and justification selected as additive factors to the base position.

The actual operation of drafting characters is determined by the set of vectors contained in the selected font, and these coordinates will make a tertiary determination of the character position relative to the **net** position selected. Most of the normal text and symbol characters employed in the supplied fonts will center the character about the **net** coordinate selected.

Select primary character font

```
(void)plotopt(SFONT, (char *)name);
```

Before any **TEXT** strings can be plotted a font must be selected. A font with a full set of printable ASCII gothic characters may be selected with **-** for **name**. The standard font library distributed with this system contains a variety of styles, alphabets and symbols. When selecting fonts from this library the first character in the name must be a **-** (eg. **-sr**). If non-library fonts are to be employed then the full path name must be specified. Note: all fonts must have been preformated in accordance with system standards (see the section on fonts).

Select alternate character font

```
(void)plotopt(SFONTA, (char *)name);
```

An alternate font may be specified with the SFONTA call. Characters in this font are selected by means of the special codes described in TEXT. If the alternate font has not been established or a character in the string does not exist in the alternate font, the corresponding character in the primary font will be drafted.

String plotting

```
(void)plotopt(TEXT, (char *)str);
```

The standard C null terminated string **str** is plotted at the current pen's x and y coordinates with the attributes ascribed to the pen's character control parameters. If the current x or y coordinates are outside of the current window then the plotting is suspended. However, the string may be positioned outside the window with appropriate XOFF or YOFF values. If any part of a character of the string is outside the plotting device's coordinate range the character will not be plotted.

Three TEXT string character values have special meaning and are used for control:

\001 select primary font for all subsequent characters,

\002 select secondary font for all subsequent characters,

\n terminate the current 'line' and continues the string with the y offset negatively adjusted. See LEAD for setting line spacing.

The font selection remains in effect for all subsequent TEXT calls for currently selected pen. The \n character only affects the string posted in the current call and will not change the basic positioning of any subsequent TEXT calls.

Character size

```
(void)plotopt(SIZE, (long)size);
```

The argument **size** is a multiplier which scales the vectors defining the character of the primary and secondary font. The basic unity scaled size of the character (height of the letter E for alphabetic fonts) is defined as 21 units high.

If **size** is negative, the absolute value of the least significant bit has a precision of $1/16^{th}$ units. For example, if the positive value of **size** is 10, then the equivalent negative value would be −160. Use of negative **size** provides greater resolution for small scaling values.

Character string rotation

```
(void)plotopt(ANG, (long)angle);
```

Set the character string rotation about the current x–y coordinates to **angle** in radians times 10,000. The angle is measured counter-clockwise from the positive x-axis. Note that x and y offset participate in the rotation (see XOFF and YOFF).

Character offsets

```
(void)plotopt(XOFF, (long)xoff);
(void)plotopt(YOFF, (long)yoff);
```

XOFF and YOFF offset the center position of a character string from the current pen x–y coordinates. Note that these offsets are also rotated by ANG.

Justification

```
(void)plotopt(JLEFT);
(void)plotopt(JRIGHT);
(void)plotopt(CENTER);
```

These operations set the justification mode of TEXT character string plotting. The default JLEFT mode specifies left justification of the text: the net coordinate position specifies the coordinate of the center of the first character of the string. Similarly, JRIGHT will preform right justification (last string character's position) and CENTER will center the string on the net coordinate.

Interline spacing

```
(void)plotopt(LEAD, (long)lead);
```

When employing the \n character in text strings the newline spacing is determined by LEAD. The value of **lead** is in $1/8^{th}$ units and is multiplied by the current size of the primary font to determine a temporary adjustment to the y-offset for each newline encountered in the TEXT string. A typical value is 12 (1.5 times the font size). If this parameter is to set overprinting will occur when a newline is encountered in text strings.

Character plotting examples

The following program demonstrates calls to several of the character string plotting options, and the graphic results are shown in figure 2. Ellipses in the listing represent initializations and setup found in the previous example.

```
    ...
/* convenient macro */
#define cmsize(s) plotopt(SIZE,\
   (long)(-(s)*16*200/21.))
   ...
main(argc, argv) char **argv; {
   float size;
   char str[20];
   long rot;
   ...
   plotopt(SFONT,"-");  /* select standard font */
   cmsize(.22);         /* set size to .22 cm   */
   moveto(.5*XMAX, YMAX); /* draw a center line */
   lineto(.5*XMAX, 0);    /* for reference      */
   moveto(.5*XMAX, .9*YMAX);
                          /* left just default  */
   plotopt(TEXT, "Left justified .22cm string");
   relmoveto(0,-100);
   plotopt(JRIGHT);     /* change justification */
   plotopt(TEXT, "Right justified string");
   relmoveto(0,-100);
   plotopt(CENTER);     /* center string        */
   plotopt(TEXT, "Centered String");
   relmoveto(0,-100);
   plotopt(JLEFT);      /* mult-line text       */
   plotopt(LEAD, (long)(12));
   plotopt(TEXT, "Some lines of text with\n\
some newlines which\n\
display a short sentence");
   moveto(.5*XMAX, .5*YMAX);
   plotopt(JRIGHT);     /* display various      */
   plotopt(XOFF, -50L); /* sizes of text        */
   for (size = .4; size > .06 ; size -= .05) {
      cmsize(size);
      sprintf(str, "%.2fcm size E", size);
      plotopt(TEXT, str);
      relmoveto(0, -80);
   }
   moveto(.75*XMAX, .25*YMAX); /* display some  */
   cmsize(.22);         /* rotated text         */
   plotopt(JLEFT);
   plotopt(XOFF, 70L);
   for (rot = 0; rot < 60000; rot += 7854) {
      plotopt(ANG, rot);
      plotopt(TEXT, "Spinner");
   }
   ...
```

Symbol Plotting

Symbol plotting posts a selected symbol at each coordinate position specified by a **pxyxmit** execution. Several of the following operations involved with symbol plotting are basically the same as for text operations except that no provision is made for offsets or string plotting. When performing symbol plotting both the font and scaling operations must be performed.

Select symbol font

```
(void)plotopt(SFONTS, (char *)name);
```

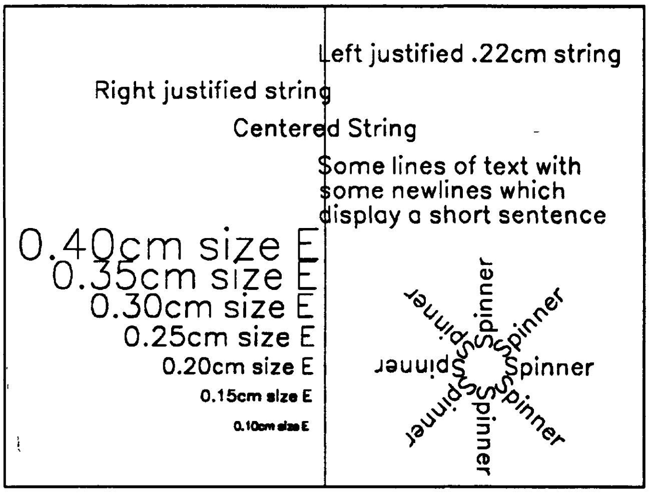

Figure 2: Example of basic string plotting calls

This operation is identical to the **SFONT** and **SFONTA** operations except that the user will tend to choose fonts that are more suitable to symbol plotting. The standard system font (name = **"-"**) may be employed for symbols since many of the characters in the range 1 to 30 are suitable for point plotting.

Symbol size

```
(void)plotopt(SSIZE, (long)size);
```

Set the scaling multiplier of the symbol character to **size**. As with character size, if **size** is negative then the absolute value of the least significant bit has a precision of 1/16 units.

Symbol character rotation

```
(void)plotopt(SANG, (long)angle);
```

Set the symbol rotation to **angle** in radians times 10,000. **Angle** is measured counterclockwise from the positive x axis.

Select symbol

```
(void)plotopt(SYM, (int)char);
```

The selected symbol will be posted at each coordinate specified by a **pxyxmit** command. If *char* = 0 then the posting operation is suspended.

Line Plotting

When a pen is initialized, the ability to draft solid lines is automatically enabled.

Dash line mode

```
(void)plotopt(DASH);
```

This option sets the drafting mode to dashed lines. Note that suitable DMASK and DSIZE selections should be made before executing a dashed line pen motion.

Dash attribute

```
(void)plotopt(DMASK, (long)mask);
```

Mask is a word which contains a bit pattern to designate the pen down (bit on) and pen up (bit off) characteristic of the dashed line. The pattern is the least 16 bits significant bits of the word.

Dash element size

```
(void)plotopt(DSIZE, (long)size);
```

Size sets the length of each bit element of the **mask**.

Select solid line mode

```
(void)plotopt(SOLID);
```

Set the line drafting mode to generate a solid line.

Select Bezier line mode

```
(void)plotopt(BEZIER)
(void)plotopt(BEZIERN)
```

Bezier line drafting generats smooth lines with two intermediate points defining the curve passing through every third node point. There should be a minimum of four points defining the curve and the total number of points −1 should be evenly divisible by 3. BEZIER turns Bezier mode on and BEZIERN turns it off.

Symbol Line Plotting

Lines may be drafted as a repeated set of symbols and optionally connected by a solid or dashed line. To deselect the symbol line option either a SOLID or DASH option must be selected.

Symbol line mode

```
(void)plotopt(FPLOT); or
(void)plotopt(FPLOTN);
```

FPLOT specifies that the symbols are joined with a line, FPLOTN specifies that symbols are not to be connected with a line.

Line Symbol Selection

```
(void)plotopt(FSYMS, (char *)str);
```

The null terminated character string **str** may contain up to 255 characters that determine the sequence of symbols to be plotted along the line. The special symbol value of \177 is reserved as a spacing character and denotes an open segment equal to the specified symbol spacing. If the most significant or sign bit of any symbol character the following character is to be rotated 180°. The font determining the symbol graphic is determined by the symbol font.

The symbol string is repeated along the line unless a \377 is encountered in **str**. In this case, character drafting is suspended until a new line is started at which point the string is drafted from the beginning. This feature is especially useful with the Bezier option in drafting curved labels.

Line Symbol Size

```
(void)plotopt(F_SIZE, (long)size);
```

Set the scaling multiplier of the symbol character to **size**. As with SIZE, if **size** is negative then the absolute value of the least significant bit has a precision of 1/16 units.

Line Symbol Separation

```
(void)plotopt(F_DIST, (long)dist);
```

Dist determines the intersymbol distance in units.

Extended line drafting examples

Figure 3 shows the results of the following example program demonstrating the extended line drafting capabilities of program **plotter**.

```
   ...
/* more useful macros */
#define cmssize(s) plotopt(SSIZE,\
   (long)(-(s)*16*200/21.))
#define cmfsize(s) plotopt(F_SIZE,\
   (long)(-(s)*16*200/21.))
   ...
   static XY
line[]={-4,-3, 2,3, 3,2, 3,1, -4,-3, T,0};
   ...
   plotopt(SFONTS, "-"); /* select standard     */
   cmssize(.2);          /* and set size        */
   plotopt(SYM, 023);    /* select symbol       */
   moveto(.25*XMAX, .75*YMAX);
   do_obj(box, 200.);    /* do box with symbol  */
   plotopt(SYM, 0);     /* turn off symbol mode */
   plotopt(DMASK, 0x5555); /* set dash mask     */
plotopt(DASH);          /* select dash mode    */
plotopt(DSIZE, 20L);    /* set dash size       */
moveto(.75*XMAX, .75*YMAX);
do_obj(box, 200.);      /* dashed box          */
plotopt(DMASK, 0xf333); /* set dash-3 dot      */
do_obj(box, 230.);      /* plot box            */
plotopt(F_DIST, 40L);   /* inter sym dist      */
cmfsize(.3);            /* set symbol line sz. */
plotopt(SFONTS, "-sym1"); /* set sym. font     */
plotopt(FSYMS, "\211\213"); /* sym string      */
plotopt(FPLOT);         /* symbol str mode     */
do_obj(box, 150.);      /* another nested box  */
plotopt(F_DIST, 0L);    /* no sym dist         */
plotopt(FPLOTN);        /* plot w/o line       */
moveto(.3*XMAX, .3*YMAX);
do_obj(line, 100.);     /* four point line     */
moveto(.5*XMAX, .3*YMAX);
plotopt(SFONTS, "-");   /* select std. font    */
plotopt(SYM, 3);        /* select symbol 3     */
plotopt(DASH);          /* set dash line mode  */
do_obj(line, 100.);     /* do line w/ symbols  */
plotopt(BEZIER);        /* Bezier line mode    */
plotopt(SOLID);         /* with solid line     */
do_obj(line, 100.);
plotopt(SYM, 0);        /* turn off symbols    */
plotopt(F_DIST, 12L);   /* add letter space    */
plotopt(FPLOTN);        /* do text on Bezier   */
plotopt(FSYMS,"Atlantic Ocean\377");
moveto(.7*XMAX, .3*YMAX);
do_obj(line, 100.);
...
```

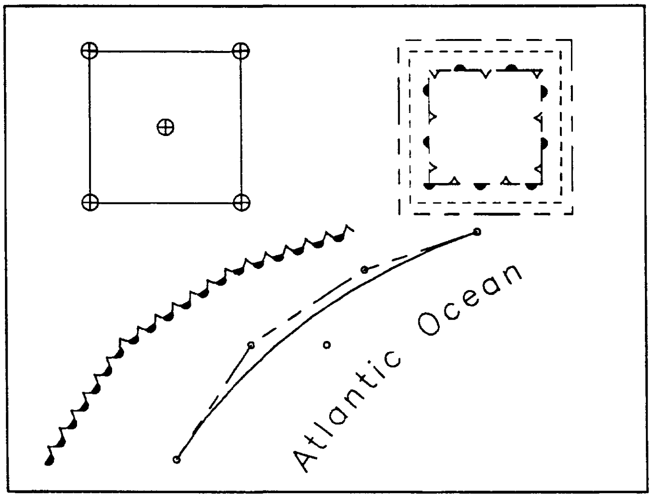

Figure 3: Examples of extended line plotting calls

Request to Plotter

The programmer can make inquiries of program **plotter** by means of the plotreq function. In all cases plotreq returns a pointer to the structure **ANSWR**:

```
#define MAX_USTR 20
typedef struct _answr {
   int cmd;
   long x, y, code;
   char str[MAX_USTR+1];
} ANSWR;
```

Depending on the option, one or more of the elements of the structure are updated by the response from **plotter**. Plotreq also ensures synchronization between the application program and **plotter**.

Return error status

```
ANSWR *p;
p = (ANSWR *)plotreq(ERROR);
```

The code value of the **ANSWR** structure is updated with the current error level of plotter. The meaning of each error code is given in *graphics.h*. Note that this call will clear the setting of error to 0 in **plotter** so that subsequent error requests will indicate error conditions since the last plotreq(ERROR) call.

Return cursor position

```
ANSWR *p;
p = (ANSWR *)plotreq(CURSOR);
```

If the graphics device is interactive and has cursor capability the current position of the cursor is returned in the x–y values of the **ANSWR** structure. In addition, if the device has supplementary information generated by the cursor, these data are placed in the character string str. If the device does not have cursor capability the command is ignored.

Return maximum x–y size of device

```
ANSWR *p;
p = (ANSWR *)plotreq(P_SIZE);
```

This function will return the maximum size of the coordinates that the device is capable of handling in the x–y structure values. Remember that the minimum x–y coordinate of any device is always considered as 0, 0.

Current font size (Archaic)

This option preceded autoscaling of the fonts and is listed here only for reference.

```
ANSWR *p;
p = (ANSWR *)plotreq(FSIZE);
p = (ANSWR *)plotreq(FSSIZE);
```

The unscaled size of the current primary pen (**FSIZE**) or secondary font (**FSSIZE**) is returned in the structure value **code**. If referenced font is not selected then a 0 value is returned.

Metagraphic file format

The metagraphic control stream for this system was designed to generate a highly compressed data stream to minimize the storage required for large volume, detailed graphic files. It is also designed to be transportable between any 8-bit byte computer processor system so that metagraphic files created on one hardware system can be displayed on a completely different system.

The structure of the metagraphic stream is quite simple, consisting of a command byte shown in Table 1 that may be followed by argument bytes. Three of the four classes of command byte have a 5 bit *option* field that allows up to 32 operations that are discussed in the executional documentation, and are mostly associated with output of the `plotopt()` procedure. *Option*'s numeric values are defined in `graphics.h`.

Table 1: Metagraphic command byte.

bits								no. arg.	
7	6	5	4	3	2	1	0	bytes	type
0	*n*		*option*					$1 \leq n \leq 3$	`_BYTE` and `_LONG`
1	0	0	*option*					none	`_NOARG`
1	0	1	*option*					variable	`_STR`
1	1	*r*	*p*	n_x		n_y		$n_x + n_y$	$\Delta x, \Delta y$ coords.

- Bits 7 and 0 are respective most- and least-significant bits.
- Character string must be null terminated (`\0` character).
- r is relative motion flag, $r = 1$ for true, $r = 0$ for false.
- p is pen-up flag, $p = 1$ for pen-up to coordinates, $p = 0$ to draw line to coordinates from current position.

The fourth class of command byte transmits the coordinates controlling pen motion along with a relative/absolute motion flag and pen control flag. Coordinate values associated with this command are in incremental mode (viz., x–y distance from the last pen position) with the initial pen position at location 0,0. Procedure `pxyxmit()` is principally associated with this command. One or both of the coordinate values may be missing if zero but if both are non-zero Δx precedes Δy in the stream.

Integer arguments associated with the `_LONG` and coordinate commands are signed values up to 3 bytes long, and the order of the output is from most significant to least significant byte. The number of required bytes is determined by the numeric value so that numbers in the range −128 to 127 require one byte, −32,768 to 32,767 require two bytes, etc.. The `_BYTE` argument is considered a single, unsigned integer with a value between 0 and 255, and `plotopt()` extracts the value from the 8 least significant bits of the second argument of type `int`,

NOTE: because a metagraphic file is a binary file, users must be sure that necessary options are selected when performing inter-system transportation of the metagraphic data.

Plotter drivers

Program **plotter** is a translator of the device independent metagraphic command stream into control operations compatible with the plotting device selected by the user at runtime. In order to maintain expansion flexibility for adding new plotters the internals of **plotter** are clearly divided into two operations: those common to all plotting operations such as parsing the metagraphic input, line clipping, character generation, etc., and those that are unique and specific to individual plotters that are contained in code sections termed device "drivers". Drivers may be either internal procedures linked and loaded as part of program **plotter** or external programs called by **plotter**.

Internal drivers

The easiest way to discuss internal drivers is to examine the debugging driver in file `Ddebug.c` (all distributed internal drivers are in files named `D`*`name`*`.c` where `D`*`name`* is the internal entry point identifier). The following is the declarative and control synopsis for `Ddebug`:

```
/* Debugging driver */
#include "plotter.h"
#include <varargs.h>
   /* "s" may be larger */
static char s[2] = "\0";
   /* required structure for return data */
static XYS cursor = {0, 0, s};
   /* basic range */
# define XPMAX  3000
# define YPMAX  2000
static long old_pen = -1;
   XYS *
Ddebug(va_alist) va_dcl {
   va_list vap;
   int cmd, i;
   long pen, x, y;
   XYS *ret = &cursor;

   va_start(vap); cmd = va_arg(vap, int);
   switch(cmd) {
   case D_SCALE: ...  /* set scaling */
   case D_INIT: ...   /* initialization, print
                         Dglobal values */
```

```
    case D_DONE: ...    /* normal completion */
    case D_PANIC: ...   /* completion due to signal
                           trap */
    case D_DISABL: ... /* disable graphics mode
                           (some terminals) */
    case D_MOVE: ...    /* move, pen-up */
    case D_LINE: ...    /* move, pen-down (draw a
                           line) */
    case D_ERASE: ...   /* erase (clear screen),
                           NO-OP on hard copy */
    case D_PEN: ...     /* select plotter's
                           mechanical pen */
    case D_STRING: ... /* device dependent str */
    case D_CURSOR: ... /* get cursor location and
                           key */
    default: ...        /* should never occur, see
                           system manager */
    }
    va_end(vap);        /* cleanup varargs */
    return(ret);
}
```

The details of each **case** statement will be discussed later.

The first item to note is that the system header file **varargs.h** and the local header file **plotter.h** (which has an embedded **stdio.h** include) must be included by any driver procedure. Program **plotter** calls the driver with one to three arguments depending upon the first or command (**cmd**) argument, and expects a return value of a pointer to a structure **XYS**:

```
typedef struct {
   long x, y;
   char *s;
} XYS;
```

for the **D_INIT** and **D_SCALE** commands where the **x** and **y** values are to be set by the driver to the maximum size of the device's plotting area (scaled by **Dglobal.scale**. In the case of interactive devices with a cursor, the returned values of **x-y** are the scaled cursor position and **s** points to a string containing the cursor key value.

Program **plotter** also maintains a structure **Dglobal** to communicate addition control information to the driver:

```
struct {
   double scale;    /* basic scaling parameter */
   int dargc;       /* count of device (-D) args */
   char *dargv[MAX_DARGS]; /* list of device (-D)
                       args */
   int reverse;     /* reverse axis */
   int model_no;    /* plotter model */
   int quiet;       /* mute <bell> wait response at
                       end of plot */
} Dglobal;
```

The only element of this structure that will change during a job is **scale** because of occurance of the metagraphic **RESCALE** command. The value of **scale** is initially set to 1.0 or the value associated with the **plotter** runline **-s** option.

Dargc and **darvg** are extracted from runline options **-D***value* where only the string *value* is addressed in **dargv**. This allows device specific options to be set at runtime.

The flags **reverse** and **quiet** are normally 0 unless the respective runline options **-r** or **-q** are used. Each device is considered to have a "normal" mode of relating the x–y axis to the respective edges of the plotting surface and **reverse** reverses this relationship. For output to interactive devices it is normally necessary to "hold" the plot on the screen until the user has had a chance to review the results. Signaling the end of the graphic operations is done either by making a cursor request which displays the cursor on the screen or by ringing the device's bell and waiting for some form of acknowledgement from the user—typically, just a "return" key stroke. In certain applications the latter operation should be suppressed with the **quiet** switch.

Many plotting devices use a common control mechanism. The Tektronix 4010–4014 control codes are examples of codes frequently employed by several other manufacturers. Rather than develop individual—nearly identical—drivers for each of these systems the differences can be accommodated by a single driver that examines the value of **model_no** to alter various phases of its operations. The initializing value of **model_no** is determined at runtime when **plotter** determines what device has been selected and looks up the selected device in a table in **devlist.c**. Driver **D4014** distributed with this system provides an example of how this parameter can be used.

Generally, output by the driver is to *stdout* which may have been **freopen**(3)'ed by **plotter** if the runline parameter **-o** was used. *Stdin* input is unaltered by **plotter**. Special manipulations of I/O control may be required, especially for hardcopy devices on RS232 ports, which may be non-portable because of variations of UNIX systems.

Opening and closing operations

Program **plotter** always calls the device driver with a **D_INIT** command before any other operations. In the case of **Ddebug** this is handled by:

```
   case D_INIT:   /* initialization, print Dglobal
                     values */
      printf("D_INIT: scale: %g, model_no: %d\n",
         Dglobal.scale, Dglobal.model_no);
      printf("\treverse: %s, quiet: %s\n",
         Dglobal.reverse ? "ON" : "OFF",
         Dglobal.quiet ? "ON" : "OFF");
      printf("\t%d -Dargs\n", Dglobal.dargc-1);
      for (i = 1; i < Dglobal.dargc; ++i)
         printf("\t\t%s\n",Dglobal.dargv[i]);
         /* return maximum size of device */
rescale:
      if (Dglobal.reverse) {
         cursor.x = YPMAX / Dglobal.scale;
         cursor.y = XPMAX / Dglobal.scale;
      } else {
         cursor.x = XPMAX / Dglobal.scale;
         cursor.y = YPMAX / Dglobal.scale;
      }
      break;
```

Since this is a debugging procedure, only the contents of the **Dglobal** is printed. The required return value for the D_INIT call is the maximum size of the plotter returned in the structure XYS's x and y values so that **plotter** can set the maximum allowable clipping window and thus ensure that **all** graphic coordinates passed to the driver will be in this range. Note that this value should be modified by **Dglobal.scale**. In the case of normal drivers other initialization code will replace the print statements. If the driver determines that it cannot function a null return will cause **plotter** to abnormally terminate.

Because of the label **rescale** in the above example, the D_SCALE will be discussed here:

```
   case D_SCALE:   /* set scaling */
      printf("D_SCALE: %g\n", Dglobal.scale);
      goto rescale;
```

This call is made only if the RESCALE metagraphic command is used and, as in the case of D_INIT, the new maximum size of the device must be returned.

Two methods of closing the driver's operation are by means of the normal D_DONE call or the abnormal D_PANIC which represents an emergency completion due to a trap on a system interrupt. In **Ddebug** these entries are handled by:

```
   case D_DONE:   /* normal completion */
      printf("D_DONE, hit return when done: ");
      if ( ! Dglobal.quiet ) {
         putchar('\006');
         fflush(stdin);
         (void) getchar();
      }
      break;
   case D_PANIC:   /* completion due to signal
                      trap */
      printf("D_PANIC\n");
      break;
```

The D_DONE code is typical of interactive devices except for the **printf** statements. Usually the **break** before **case D_PANIC** is omitted and "cleanup" code common to both entries is placed in the D_PANIC section. If **plotter** is installed, the reader may simply execute:

```
plotter -d debug
```

The D_INIT printout will appear on the terminal and **plotter** will expect metagraphic input from the keyboard. Entering a ^D (end of file) causes the D_DONE messages to appear. Striking the interrupt key will cause the D_PANIC entry to be called.

Selecting mechanical pen

After the initialization call and before any drafting operations **plotter** always calls the driver with a D_PEN command where the second argument is type long pen number:

```
   case D_PEN:   /* select plotter's mechanical
                    pen */
      pen = va_arg(vap, long);
      printf("D_PEN: %ld (replacing: %ld)\n",
         pen, old_pen);
      old_pen = pen;
      break;
```

Because **plotter** is often repetitive with this call the driver program should retain the value of the **last** pen call and ignore the entry when the last and current pen are identical.

Because a metagraphic integer is at most three bytes long the actual length of pen is up to 23 bits (8388608). Actual translation of the mechanical pen number into physical pens or other line drafting attributes available on the device is a decision of the driver writer. In most cases of plotters with **no_pens** true mechanical pens the author has determined the final selected pen by: **pen %= no_pens**.

Drafting operations

Drafting operations are controlled by the D_MOVE and D_LINE commands where there are two, type long coordinates in the argument list:

```
   case D_MOVE:   /* move, pen-up */
      printf("D_MOVE");
      goto printxy;
   case D_LINE: /* move, pen-down (draw a line) */
      printf("D_LINE");
printxy:
      if (Dglobal.reverse) {
         y = va_arg(vap, long) *
             Dglobal.scale + 0.5;
         x = XPMAX - va_arg(vap, long) *
             Dglobal.scale + 0.5;
      } else {
         x = va_arg(vap, long) *
             Dglobal.scale + 0.5;
         y = va_arg(vap, long) *
             Dglobal.scale + 0.5;
      }
```

```
    printf(" x/y: %6ld %6ld\n", x, y);
    break;
```

The scaled values of **x** and **y** will always be in the range from 0 to the maximum limits returned to **plotter** in response to the `D_INIT` or `D_SCALE` calls. Computations required for axis reversal often vary with capabilities of the device.

Cursor input

Cursor input is designed principally for use with interactive software for tty type terminals such as MAPGEN's and PLOTGEN's **zoom** or **preview** program. The `D_CURSOR` command entry (no additional arguments) expects the driver to return the current position of the cursor and a keyboard character simulated by the **Ddebug** driver as:

```
case D_CURSOR:    /* get cursor location and
                     key */
    for(;;) {
        printf("D_CURSOR: enter x y c\n");
        fflush(stdin);  /* no type-ahead */
        if (scanf("%ld %ld %1s",&cursor.x,
            &cursor.y, s) == 3) break;
        printf("?\n");
    }
    if (Dglobal.reverse) {
        cursor.y = cursor.x / Dglobal.scale + .5;
        cursor.x = cursor.y / Dglobal.scale + .5;
    } else {
        cursor.x = cursor.x / Dglobal.scale + .5;
        cursor.y = cursor.y / Dglobal.scale + .5;
    }
    break;
```

Miscellaneous control

Two commands, `D_DISABL` and `D_ERASE`, are similar in usage to the `D_CURSOR` command and are used in interactive applications. `D_DISABL` is used with terminals that have two modes of screen display: text and graphics. The driver is expected to put automatically the terminal in graphics mode in response to graphic calls and return it to text mode only when a `D_DONE`, `D_PANIC` or `D_DISABL` command is executed. `D_ERASE` is a graphic command that should clear the terminal's screen.

Note that the `D_CURSOR`, `D_ERASE` and `D_DISABL` commands are normally ignored by hardcopy output devices.

Via `plotopt(SPECIAL,)`, the `D_STRING` command allows device dependent information to be passed to the driver through the metagraphic stream. The second argument of this entry is a pointer to a null terminated string. Usage of this option eliminates the device independence of the system, but it may be necessary for some applications. The `SPECIAL-D_STRING` is transparent to **plotter**'s operation.

Code for `Ddebug.c`'s version of the miscellaneous commands are:

```
case D_DISABL:   /* disable graphics mode (some
                    terminals) */
    printf("D_DISABL\n");
    break;
case D_ERASE:   /* erase (clear screen), NO-OP
                   on hard copy */
    printf("D_ERASE\n");
    break;
case D_STRING:  /* device dependent string */
    printf("D_STRING: <%s>\n",
            va_arg(vap, char *));
    break;
```

Example execution of Ddebug driver

A test program **boxes(1)** is distributed with the system which generates a set of nested boxes where each box is drawn with a new mechanical pen. To show **plotter**'s operation with the internal debug driver execute:

```
boxes 10 2 1 -d debug
```

The following should appear on the terminal:

```
D_INIT: scale: 1, model_no: 0
   reverse: OFF, quiet: OFF
   0 -Dargs
D_ERASE
D_PEN: 0 (replacing: -1)
D_STRING: <Boxes special test>
D_PEN: 0 (replacing: 0)
D_MOVE x/y:      0      0
D_LINE x/y:     30      0
D_LINE x/y:     30     30
D_LINE x/y:      0     30
D_LINE x/y:      0      0
D_PEN: 1 (replacing: 0)
D_MOVE x/y:     10     10
D_LINE x/y:     20     10
D_LINE x/y:     20     20
D_LINE x/y:     10     20
D_LINE x/y:     10     10
D_DONE, hit return when done:
```

External drivers

When the author first became involved with developing programs for operating plotters in the mid 1960's it was relatively simple to develop the sequence of commands necessary to control their actions. Even into the early 1980's—including the intervening advent of graphic terminals—most machines could be controlled with code that was not difficult to develop. But as technology developed, the sophistication of the graphic machines dramatically increased and caused a significant increase in control complexity. Because of this complexity, many manufacturers recognized that they needed to supply software (often as proprietary source code) with their hardware product for use by the buyer's application graphic packages. An example of such software is the Calcomp FORTRAN graphic subroutine library supplemented with additional entries allowing access to features unique to the manufacturer's device.

Including such proprietary software as an integral part of program **plotter** would limit its distribution and multiple systems using identically named procedures could not be linked into one program. To accommodate this situation the external driver was introduced where external programs unique to each device in the local environment would be spawned by **plotter** and passed the basic graphic operations decoded from the metagraphic input.

Internal driver procedure **Dextdev** provides the mechanism allowing **plotter** to execute and control an external driver via the control codes in Table 2. In this case, the **Dglobal.model** number defines the external device program to be executed and whose name and size characteristics are extracted from the structure array **model**:

```
...
# include <signal.h>
# include <varargs.h>
# include "plotter.h"
#define EXTDEV
#define PLOTTER
# include "graphics.h"
#define CTS_CM   200
...
struct {
   char   *prog;         /* name of external
                            program */
   long   xsize, ysize; /* size of plotter */
} model[] = {
   "extdebug", 15000, 10000, /* simple listing
                                routine */
   "c1077", 65535, 17525,    /* Calcomp 1077 */
   ...
   "dp8", 65535, 17270,      /* Houston DP8 */
   (char *)0,910,700,        /* Zoom version of
                                SunCore */
};
# define XPMAX model[Dglobal.model_no].xsize
# define YPMAX model[Dglobal.model_no].ysize
```

In the cases of hard copy plotters the x and y sizes represent 200 counts/cm devices and final scaling will be done by the external driver when necessary. Note that **EXTDEV** and **PLOTTER** must be defined before the included header **graphics.h** to resolve the control code acronyms appearing in Table 2. The external driver **extdebug** is supplied with the system for testing purposes.

Table 2: External driver communication stream.

Operation	Code	Argument bytes			
Beginning of plot	_BOP	none			
End of plot	_EOP	none			
Select pen (short)	_PEN	p_0			
Select pen (24 bit)	_PENL	p_2	p_1	p_0	
Move pen-up	_MOVE	x_1	x_0	y_1	y_0
Draw to	_DRAW				
Special string	_SPCL	c_0	...	c_n	0

The subscript numbers for p, x and y denote their significance. Pen motion coordinates are unsigned, absolute positions.

The driver uses the **model** information in the initialization phase in spawning the external driver program:

```
...
case D_INIT:
   ...
   if (model[Dglobal.model_no].prog) {
      strcpy(name, getenv(_GENVB));
      *(strrchr(name, '/') + 1) = '\0';
      strcat(name,
         model[Dglobal.model_no].prog);
      Dglobal.dargv[0] = name;
      sprintf(ssize,"-S%d,%d,%d",XPMAX,YPMAX,
         CTS_CM);
      Dglobal.dargv[Dglobal.dargc] = ssize;
      if (!(translat = wpopen(Dglobal.dargv)))
         bomb(1,"ext. driver %s failed to\
exec\n",name);
      (void)signal(SYS_SIGCLD, dead_child);
   } else
      translat = stdout;
   putc(_BOP, translat);
...
```

The **Dglobal.dargv** array is used as the system procedure **execv**(2)'s arguments so that runline **-D** arguments as well as the plotter size defined in the local **model** table and the resolution of the data in counts/cm in a **-S** option are passed to the external driver Note that the **-D** part is missing from the former arguments. Full path name of the external driver requires that the external driver must reside in the same directory as program **plotter** which is determined from the environment variable set by the compile time macro **_GENVB** (typically **"GRAPHB"**).

The **popen**(3) UNIX function was not used because it requires a string equivalent to a keyboard execution and performs an intermediate execution of **sh**(1) with this string. Procedure **wpopen** (see file **wpopen.c** for details) directly spawns the external driver specified in the first element of the string pointer array argument and passes the entire array as arguments for the **argv** of the spawned process **main** entry. *Stdout* and *stderr* of the external driver will be the same as for **plotter** and *stdin* will be connected to the pipe from the local *stream* pointer **translat**. As an equivalent of **pclose**(3), procedure **wpclose** (in file **wpopen.c**) is called in the **D_DONE–D_PANIC** section of the driver.

If the first argument of the external procedure **bomb** (in file **plotter.c**) is non-zero it indicates that a system error is encountered and procedure **perror**(3) should be called. The remainder of the arguments are passed to **vfprintf**(3) as format and format arguments. **Bomb** does not return.

For conditions where the external driver is not found or fails **signal**(2) is used to trap to the local function **dead_child** and thus prevent **plotter** from continuing to write data to a non-existant process. Unfortunately, this monitoring of a spawned process currently varies between different versions of the UNIX operating system. The conditional compile statements at the beginning of the file **Dextdev.c** attempt to resolve some of these differences.

Use of plotter with graphical interface systems

Previous graphic terminal applications made the development of a single, interactive program working in a bidirection communications mode with **plotter** a simple and effective method of operation. In this case, it was **plotter**'s responsibility to provide interface with the terminal and not the interactive, calling program, thus one program could support a variety of terminal types. But recent developments in UNIX graphical interface subsystems such as X11 has caused a change in how **plotter** is to be used in these environments.

In the case of X11, the interactive program is necessarly limited to the X11 environment which already has basic mechanisms for doing vector graphic primatives and, consequently, can perform as the device driver for **plotter**. Two implementation methods are possible:

1. revise **plotter** into a procedure to be directly linked in to the application program, or

2. spawn **plotter** as a process which outputs back to the parent process in a manner similar to using external drivers.

The latter method was chosen to avoid problems such as ensuring reenterability of the procedures and other basic recoding and debugging problems.

Dextdev provides this capability when **model[].prog** is null and it is assumed that the output will go to *stdout*. Thus, a program executing progam **plotter** has only to execute **plotter** and convert its output data into vector graphic commands required by the graphical interface system. Although this **model** entry may have predefined coordinate limits, these limits can be readily changed by use of the **-DR** runline argument and thus can be used by by any calling system.

Installing device drivers

Drivers installed during compilation and linkage of program **plotter** are determined by the file **devlist.c**. This file contains two parts: necessary declarations of drivers in the system and a structure array **dev_list**. Because it is normal for most compilers not to require that declared external references be resolved unless they are part of executable code or data initializations, the declaration list may contain modules not linked into the final program.

The list of drivers to be linked into **plotter** is contained in the structure declared in **plotter.h**:

```
struct DEV_LIST {
   char *name;    /* device name referenced by -d
                     option or TERM environment
                     parameter */
   XY *(*dev)();    /* device dependent driver
                       routine */
   int model;    /* Device options for small
                    variations in generic device.
                    See individual drivers for
                    meaning */
};
```

Name is how the driver is referred to at execution time by either explicit use of the **-d** runline parameter or the environment parameters **GTERM** or **TERM**. Structure elements **dev** and **model** are pointers to the driver's entry point and previously discussed model number.

In **devlist.c** the global list of devices **dev_list** is initialized as shown in the following example:

```
   struct DEV_LIST
dev_list[] = {
      /* should leave these here */
   "debug",     Ddebug,   0, /* debug device mode*/
   "exdebug",   Dextdev,  0, /* debug external
                                device mode */
   "ranger",    Dranger,  0, /* range determination
                                routine */
      /* selected for local system needs */
   "ega",       Dgfx,     0, /* uPort gfx */
   "AT386",     Dgfx,     0, /* uPort gfx */
   "at386",     Dgfx,     0, /* uPort gfx */
   "ps",        Dpstscr,  0, /* PostScript
                                printer */
   "go140",    D4014,    2, /* GraphOn 140 */
   "c5800",    Dextdev,  5, /* Calcomp 5800
                               electrostatic */
   "sunzoom",  Dextdev,  7, /* SunCore - zoom */
   ...
#ifdef IGNORE
   "c970",     Dc970,    0, /* Calcomp 970
                               (200 c/cm res) */
   "kong",     Dkong,    0, /* Kongsburg -
                               photohead */
   ...
   "imogen",   Dextdev,  4, /* Imogen laser */
# endif
   "bumdev",   Dbumdev,  0,
   0,       0,    0,
};
```

Entries in the conditional compilation section (always false) accommodates a complete list of available drivers but are not linked in the current installation. The last two lines should remain unaltered because **plotter** senses the null name field of the last line as the end of the list and uses the last entry, `Dbumdev`, as a default driver.

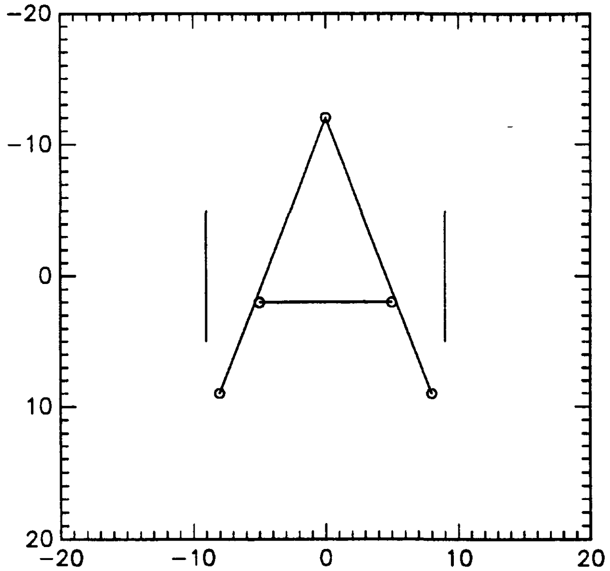

Figure 4: Stroke pattern for the Hershey Simplex Roman letter A, character 501.

Fonts

Characters and symbols in this graphic system are drafted or "stroked" characters with most of the characters based upon the Hershey symbol coordinate tables (Wolcott and Hilsenrath, 1976). For use by program **plotter**, selected characters are selected and encoded into binary font files to be loaded at runtime. Appendix A contains tables of the standard set of font symbols distributed with the graphics system. In this section the graphic layout of the Hershey system, character table files, and generation of fonts for use by **plotter** are discussed.

Hershey Drafting System

Because it is often necessary to add characters or symbols to the graphic system a short review of the mechanism will be made. A Hershey character is based upon an integral cartesian grid with positive y–axis down the page. Original limits to the axis were ±49 units but this has been expanded to ±127 units which is sufficient resolution for most graphic applications using stroked characters. Unlike modern computer typographic descriptive methods the character coordinates are centered (or nearly so) on the coordinate system rather than being relative to a baseline and left edge.

Figure 4 shows the detail of drafting the letter A where the small circles are at the line node points and the two vertical bars denote the horizontal size of the character. The Hershey table data for drafting this character are in the form:

```
501:-9 9:0 -12:-8 9:128 0:0 -12:8 9:128 0:-5 2:
:5 2:129 0:
```

The first number is the Hershey character number which is followed by the x coordinates denoting the horizontal limits of the character and the remaining numeric pairs are either x–y coordinate pairs or pen control. Original pen control was special coordinates `-64 0` for penup to the next coordinate and `-64 -64` to terminate the character but this was altered by the author to respective `128 0` and `129 0` so as to not conflict with the new ±127 count resolution. A colon character must separate the fields and begin a continuation line. Figure 5 shows a roman A which is drafted from the table entry:

```
3001:-10 10:0 -12:-7 8:128 0:-1 -9:5 9:128 0:0 -9:
:6 9:128 0:0 -12:7 9:128 0:-5 3:4 3:128 0:-9 9:
:-3 9:128 0:2 9:9 9:128 0:-7 8:-8 9:128 0:-7 8:
:-5 9:128 0:5 8:3 9:128 0:5 7:4 9:128 0:6 7:8 9:
:129 0:
```

The Hershey notation for character styles does not follow common typological usage. Characters referred to as "simplex" roman are normally termed gothic (i.e. even weighting on all lines). Only those characters termed "complex" or "triplex" roman are truly roman. Similarly, Hershey's "gothic" are normally referred to as text faces.

The original Hershey data set, updated with the characters from pages 24 and 25 of Wolcott and Hilsenrath

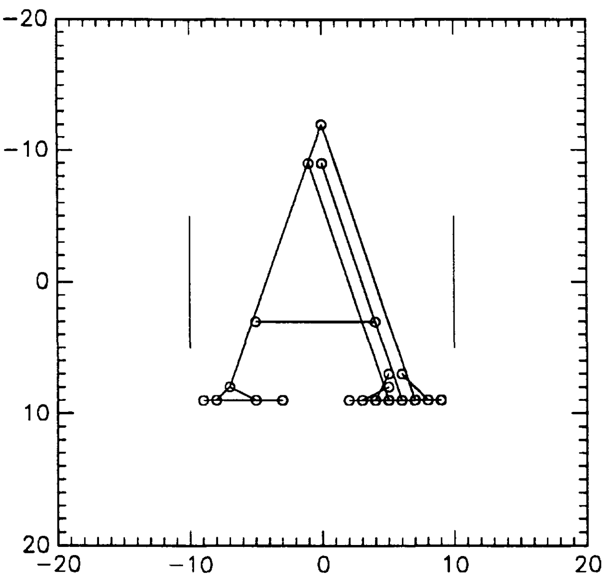

Figure 5: Stroke pattern for the Hershey Triplex Roman letter A, character 3001.

(1976), consists of 1,595 characters numbered in monotonically ascending order from 1 to 3926 and are contained in the file `H0000.tab.Z`. Additional characters have been added by the author and others, and are contained in files `H4000.tab` and up. Although there are gaps in the number sequence of the original set the author has only added characters with numbers above 4000 and has used file names which "echo" with the correct sort order.

Font structure

Program **plotter** requires characters from the Hershey tables to be selected and placed into 128 character, binary font files whose structure is:

```
struct {
   unsigned short size;       /* total byte size of
                                 structure */
   unsigned short dir[128]; /* offset to start of
                                 coordinates in
                                 vect */
   signed char vect[];        /* x-y pairs of
                                 coordinates */
} font;
```

Character coordinates are extracted from the structure by:

```
    unsigned offset, ...
    char *ptr, ...
    int x, y;

    if (offset = font.dir[in_char & 0x7f]) {
        ptr = vect + offset;
        x = *ptr++;
        y = *ptr++;
        ...
    } /* else no character in position */
```

Pen-up code of the Hershey tables is changed to a −128,0 x–y value and the end of character is a −128,−128 value.

The vertical size of a font is stored in `font.vect[0]`, (unused because a zero or null character terminates the string) and allows **plotter** to normalize all fonts to a common size of 21 units. For fonts containing the alphabet, size is the height of the letter E and, for purely symbolic fonts, it is determined by general symbol size.

To create the fonts files the program **symgen** is provided which interprets a control file selecting the Hershey characters by Hershey number and assigns them to the `font.dir` position. Execution of this program is:

```
symgen select_file font_file Hershey_file[s]
```

The first runline argument is the name of a file which contains a mapping of the selected characters and their position in the font. The second argument is the name of the output font file for **plotter** followed by the set of Hershey data files in ascending Hershey number order.

The first line of the selection file contains two numeric entries: vertical size of the font and baseline offset. This is followed by lines containing either a single ASCII character or a two or three digit number followed by the Hershey symbol number. Anything else on the line is considered a comment. Table 3 in Appendix A lists the contents of file `sr.D` used to create font `-sr` shown in table 21. The order of the entries in the selection file is not critical because **symgen** sorts the selection entries by Hershey number so that the Hershey tables need be scanned only once.

All of the standard font selection files use the naming convention of `font.D` and are stored in the directory *basic-fonts*. By convention, local or test font selection files are suffixed with **.d**. A script file `Makefonts` is provided to generate all of the standard fonts used by the system as well as any local .d fonts.

Typesetting fonts

For those systems which have device independent **troff** (occasionally called **dtroff** or **ditroff**) special fonts are also supplied as well as software (**plroff**) to interpret the output and convert it to a system metagraphic stream. This requires the UNIX utility **makedev** to process the character set descriptor tables in directory `troff` which are to be installed in `/usr/lib/font/devGRAPH`.

The fonts associated with typesetting have the origin of the axis shifted to the left x size position (first numeric value after the Hershey number) and the baseline indicated by the second argument on the first line of the selection file. Tables 26–30 in Appendix A display these fonts.

References

Evenden, G.I., Botbol, J.M., 1985, User's manual for MAPGEN (UNIX version): a method to transform digital cartographic data to a map: U.S. Geological Survey Open-File Report 86-706, 134 p.

Wolcott, N.M., Hilsenrath, 1976, Tables of Coordinates for Hershey's Repertory of Occidental Type Fonts and Graphic Symbols: NBS Special Publication 424.

Appendix A—Standard Fonts

The standard fonts for this graphic system are listed in this appendix and, to ensure portability of overlay files, should be installed without modification. If new fonts are desired additional entries should be added by local site managers. Table 3 generates the standard font **-sr** that is shown in table 21 is an example of a control file used by program **symgen** to create font files for program **plotter**.

For sites without device independent **troff**, fonts prefixed with **nr** (tables 26–30) may be removed.

Table 3: Contents of **symgen** selection file `sr.D` for making **plotter** font file **-sr**.

21	0	32	0698	@	2273	`	2249
01	899	!	0714	A	0501	a	0601
02	900	"	0717	B	0502	b	0602
03	901	#	0733	C	0503	c	0603
04	904	$	0719	D	0504	d	0604
05	2281	%	2271	E	0505	e	0605
06	841	&	0734	F	0506	f	0606
07	842	'	0716	G	0507	g	0607
08	843	(	0721	H	0508	h	0608
09	844	)	0722	I	0509	i	0609
10	845	*	0728	J	0510	j	0610
11	846	+	0725	K	0511	k	0611
12	847	,	0711	L	0512	l	0612
13	850	-	0724	M	0513	m	0613
14	851	.	0710	N	0514	n	0614
15	852	/	0720	O	0515	o	0615
16	856	0	0700	P	0516	p	0616
17	857	1	0701	Q	0517	q	0617
18	862	2	0702	R	0518	r	0618
19	2284	3	0703	S	0519	s	0619
20	2293	4	0704	T	0520	t	0620
21	735	5	0705	U	0521	u	0621
22	727	6	0706	V	0522	v	0622
23	2263	7	0707	W	0523	w	0623
24	2266	8	0708	X	0524	x	0624
25	2278	9	0709	Y	0525	y	0625
26	2276	:	0712	Z	0526	z	0626
27	2277	;	0713	[	2223	{	1407
28	2233	<	2241	\	4000	|	723
29	766	=	0726	]	2224	}	1408
30	642	>	2242	^	2247	~	2246
31	0718	?	0715	_	4001		

Table 4: Symbols for font `-3osw`

	0	1	2	3	4	5	6	7
\00x				≠	√	÷	'	'
\01x				×			·	•
\02x					Ä	Ö	Ü	ä
\03x	ö	ü	ß	≥	≤			→
\04x		!	"	#	$	%	&	'
\05x	(	)	°	+	,	-	.	/
\06x	0	1	2	3	4	5	6	7
\07x	8	9	:	;	<	=	>	?
\10x		A	B	C	D	E	F	G
\11x	H	I	J	K	L	M	N	O
\12x	P	Q	R	S	T	U	V	W
\13x	X	Y	Z	[	\	]	^	_
\14x	`	a	b	c	d	e	f	g
\15x	h	i	j	k	l	m	n	o
\16x	p	q	r	s	t	u	v	w
\17x	x	y	z	{	|	}	~	

Table 5: Symbols for Cartographic font (`-cartr`).

	0	1	2	3	4	5	6	7
\00x		○	□	△	◇	☆	+	×
\01x	*	●	■	▲	★			
\02x								
\03x				·	×	→		°
\04x		!	"	#	$		&	'
\05x	(	)	*	+	,	−	.	/
\06x	0	1	2	3	4	5	6	7
\07x	8	9	:	;		=		?
\10x		A	B	C	D	E	F	G
\11x	H	I	J	K	L	M	N	O
\12x	P	Q	R	S	T	U	V	W
\13x	X	Y	Z					
\14x		A	B	C	D	E	F	G
\15x	H	I	J	K	L	M	N	O
\16x	P	Q	R	S	T	U	V	W
\17x	X	Y	Z		|			

Table 6: Symbols for Cyrillic font (-cc).

	0	1	2	3	4	5	6	7
\00x								
\01x								
\02x								
\03x								
\04x		ю	я					
\05x								
\06x								
\07x								
\10x		А	Б	В	Г	Д	Е	Ж
\11x	З	И	Й	К	Л	М	Н	О
\12x	П	Р	С	Т	У	Ф	Х	Ц
\13x	Ч	Ш	Щ	Ъ	Ы	Ь	Э	Ю
\14x	Я	а	б	в	г	д	е	ж
\15x	з	и	й	к	л	м	н	о
\16x	п	р	с	т	у	ф	х	ц
\17x	ч	ш	щ	ъ	ы	ь	э	

Table 7: Symbols for font (-cgi).

	0	1	2	3	4	5	6	7
\00x								
\01x								
\02x								
\03x								
\04x								
\05x								
\06x								
\07x								
\10x		Α	Β	Γ	Δ	Ε	Ζ	Η
\11x	Θ	Ι	Κ	Λ	Μ	Ν	Ξ	Ο
\12x	Π	Ρ	Σ	Φ	Χ	Ψ		
\13x								
\14x		α	β	γ	δ	ε	ζ	η
\15x	ϑ	ι	κ	λ	μ	ν	ξ	o
\16x	π	ρ	σ	φ	χ	ψ		
\17x								

Table 8: Symbols for font -**cip**

	0	1	2	3	4	5	6	7
\00x								
\01x								
\02x								
\03x			*ff*	*fi*	*fl*	*ffi*	*ffl*	
\04x								
\05x								
\06x								
\07x								
\10x		*A*	*B*	*C*	*D*	*E*	*F*	*C*
\11x	*H*	*I*	*J*	*K*	*L*	*M*	*N*	*O*
\12x	*P*	*Q*	*R*	*S*	*T*	*U*	*V*	*W*
\13x	*X*	*Y*	*Z*					
\14x		*a*	*b*	*c*	*d*	*e*	*f*	*g*
\15x	*h*	*i*	*j*	*k*	*l*	*m*	*n*	*o*
\16x	*p*	*q*	*r*	*s*	*t*	*u*	*v*	*w*
\17x	*x*	*y*	*z*					

Table 9: Symbols for font -**cri**

	0	1	2	3	4	5	6	7
\00x								
\01x								
\02x								
\03x								°
\04x		!	"	#	$	%	&	'
\05x	(	)	*	+	,	−	.	/
\06x	0	1	2	3	4	5	6	7
\07x	8	9	:	;	<	=	>	?
\10x	@	A	B	C	D	E	F	G
\11x	H	I	J	K	L	M	N	O
\12x	P	Q	R	S	T	U	V	W
\13x	X	Y	Z	[	\	]	^	∮
\14x	`	a	b	c	d	e	f	g
\15x	h	i	j	k	l	m	n	o
\16x	p	q	r	s	t	u	v	w
\17x	x	y	z	{	|	}	~	

Table 10: Symbols for font -crp

	0	1	2	3	4	5	6	7
\00x								
\01x								
\02x								
\03x			ff	fi	fl	ffi	ffl	°
\04x		!	"	#	$	%	&	'
\05x	(	)	*	+	,	−	.	/
\06x	0	1	2	3	4	5	6	7
\07x	8	9	:	;	<	=	>	?
\10x	@	A	B	C	D	E	F	G
\11x	H	I	J	K	L	M	N	O
\12x	P	Q	R	S	T	U	V	W
\13x	X	Y	Z	[	\	]	^	_
\14x	`	a	b	c	d	e	f	g
\15x	h	i	j	k	l	m	n	o
\16x	p	q	r	s	t	u	v	w
\17x	x	y	z	{	|	}	~	

Table 11: Symbols for font -cscp

	0	1	2	3	4	5	6	7
\00x								
\01x								
\02x								
\03x								°
\04x		!	"		$		&	'
\05x	(	)	*	+	,	−	.	/
\06x	0	1	2	3	4	5	6	7
\07x	8	9	:	;		=		?
\10x		A	B	C	D	E	F	G
\11x	H	I	J	K	L	M	N	O
\12x	P	Q	R	S	T	U	V	W
\13x	X	Y	Z					
\14x	'	a	b	c	d	e	f	g
\15x	h	i	j	k	l	m	n	o
\16x	p	q	r	s	t	u	v	w
\17x	x	y	z					

Table 12: Symbols for font **-din**

	0	1	2	3	4	5	6	7
\00x								
\01x								
\02x					Ä	Ö	Ü	ä
\03x	ö	ü	ß	≥	≤			→
\04x		!			$	%		
\05x	(	)	*	+	,	-	.	/
\06x	0	1	2	3	4	5	6	7
\07x	8	9	:		<	=	>	
\10x		A	B	C	D	E	F	G
\11x	H	I	J	K	L	M	N	O
\12x	P	Q	R	S	T	U	V	W
\13x	X	Y	Z	[	\	]		_
\14x		a	b	c	d	e	f	g
\15x	h	i	j	k	l	m	n	o
\16x	p	q	r	s	t	u	v	w
\17x	x	y	z	{	|	}		

Table 13: Symbols for font **-dr**

	0	1	2	3	4	5	6	7
\00x		·	•	∘	○	⊙	□	△
\01x	◊	☆	+	×	∗	■	■	▲
\02x	★	⚑	⚒	⊕	✱	⌑	×	←
\03x	∇	‡	§	†	±	∞	π	°
\04x		!	"	#	$	%	&	'
\05x	(	)	*	+	,	−	.	/
\06x	0	1	2	3	4	5	6	7
\07x	8	9	:	;	<	=	>	?
\10x	@	A	B	C	D	E	F	G
\11x	H	I	J	K	L	M	N	O
\12x	P	Q	R	S	T	U	V	W
\13x	X	Y	Z	[	\	]	^	_
\14x	`	a	b	c	d	e	f	g
\15x	h	i	j	k	l	m	n	o
\16x	p	q	r	s	t	u	v	w
\17x	x	y	z	{	|	}	~	

Table 14: Symbols for font **-engl**

	0	1	2	3	4	5	6	7
\00x								
\01x								
\02x								
\03x								°
\04x		!	"		$		&	'
\05x	(	)	*	+	,	–	.	/
\06x	0	1	2	3	4	5	6	7
\07x	8	9	:	;		=		?
\10x		A	B	C	D	E	F	G
\11x	H	I	J	K	L	M	N	O
\12x	P	Q	R	S	T	U	V	W
\13x	X	Y	Z					
\14x	‘	a	b	c	d	e	f	g
\15x	h	i	j	k	l	m	n	o
\16x	p	q	r	s	t	u	v	w
\17x	x	y	z					

Table 15: Symbols for font **-germ**

	0	1	2	3	4	5	6	7
\00x								
\01x								
\02x								
\03x								
\04x								
\05x								
\06x								
\07x								
\10x		A	B	C	D	E	F	G
\11x	H	I	J	K	L	M	N	O
\12x	P	Q	R	S	T	U	V	W
\13x	X	Y	Z					
\14x		a	b	c	d	e	f	g
\15x	h	i	j	k	l	m	n	o
\16x	p	q	r	ſ	t	u	v	w
\17x	x	y	z	s	ß	tz		

Table 16: Symbols for font **`-ital`**

	0	1	2	3	4	5	6	7
\00x								
\01x								
\02x								
\03x								
\04x								
\05x								
\06x								
\07x								
\10x		A	B	C	D	E	F	G
\11x	H	I	J	K	L	M	N	O
\12x	P	Q	R	S	T	U	V	W
\13x	X	Y	Z					
\14x		a	b	c	d	e	f	g
\15x	h	i	j	k	l	m	n	o
\16x	p	q	r	s	t	u	v	w
\17x	x	y	z					

Table 17: Symbols for font **`-ksym1`**

	0	1	2	3	4	5	6	7
\00x		▤	▦	▩	▦	■	━	━
\01x	━	━	┅	╍	━	━	⇥	T
\02x	⚓	ʃ	○	✕	▭	∟	┘	ѱ
\03x	ℓ	♀	♀	↑	Λ	Ω	^	⊣⊢
\04x	⎍	∩	▜	⌗	●	Q	◡	~
\05x	☘	■	⊜	◉	◎	⊙	∘	◬
\06x	◯	⌂	∟	⛪	♁	♉	●	⚲
\07x	⎍	t	ƭ	⛺	Ī	▵	ô	Ī
\10x	ȯ	♗	◺	◼	♀	🅿	⍋	Ⓣ
\11x	Ⓗ	🄷	ɗ	▲	▲	⏢	♖	σ
\12x	⯁	✈	♪	∴	≍	×	+	⚒
\13x	◓	◒	≻	Ẍ	Ĭ	⌓	Ω	ȯ
\14x	ō	♉	8	🔥				
\15x								
\16x								
\17x								

Table 18: Symbols for font **-ksym2**

	0	1	2	3	4	5	6	7
\00x		+	×	○	⊕	△	□	●
\01x	—	|	·	⁄	⟍	/	⟍	//
\02x	▧	▩	▩	⊛	▼	▲	⊓⊓	⊔⊔
\03x	•••	••	•	◔	+	⊛	/	//
\04x	//	///	▩	■	■	⊗	◉	▲
\05x	⧄	⧄	▨	▨	▩	▩	▩	▮
\06x	▬	✡	▭	▣	▫	▫▫	▫▫	◫
\07x	|	⧄⧄	//	▦	▦	▬	▬	▨
\10x	///	▩	▩	∴	▴▴▴	▱	▰	◤
\11x	◢	⧄	⁄⁄	▨	⁄⁄	⊡	⚒	═━
\12x	\	\\	\\\	▧	▩	■	■	·
\13x	⟍	⁘	◓	◒	◑	◐	-	▵
\14x	◂	▵	⧄					
\15x								
\16x								
\17x								

Table 19: Symbols for font **-osw**

	0	1	2	3	4	5	6	7
\00x				≢	√	÷	'	‘
\01x				×			.	•
\02x					Ä	Ö	Ü	ä
\03x	ö	ü	ß	≥	≤			⇒
\04x		!	"	#	$	%	&	'
\05x	(	)	o	+	.	-	.	/
\06x	0	1	2	3	4	5	6	7
\07x	8	9	:	;	<	=	>	?
\10x		A	B	C	D	E	F	G
\11x	H	I	J	K	L	M	N	O
\12x	P	Q	R	S	T	U	V	W
\13x	X	Y	Z	[	\	]	^	_
\14x	`	a	b	c	d	e	f	g
\15x	h	i	j	k	l	m	n	o
\16x	p	q	r	s	t	u	v	w
\17x	x	y	z	{	|	}	~	

Table 20: Symbols for font **-sg**

	0	1	2	3	4	5	6	7
\00x								
\01x								
\02x								
\03x								
\04x								
\05x								
\06x								
\07x								
\10x		Α	Β	Γ	Δ	Ε	Ζ	Η
\11x	Θ	Ι	Κ	Λ	Μ	Ν	Ξ	Ο
\12x	Π	Ρ	Σ	Φ	Χ	Ψ		
\13x								
\14x		α	β	γ	δ	ε	ζ	η
\15x	ϑ	ι	κ	λ	μ	ν	ξ	ο
\16x	π	ρ	σ	φ	χ	ψ		
\17x								

Table 21: Symbols for font **-sr**

	0	1	2	3	4	5	6	7
\00x		·	•	∘	○	⊙	□	△
\01x	◊	☆	+	×	*	●	■	▲
\02x	★	⚑	⚒	⊕	✱	⌑	×	←
\03x	∇	‡	§	†	±	∞	π	°
\04x		!	"	#	$	%	&	'
\05x	(	)	*	+	,	−	.	/
\06x	0	1	2	3	4	5	6	7
\07x	8	9	:	;	<	=	>	?
\10x	@	A	B	C	D	E	F	G
\11x	H	I	J	K	L	M	N	O
\12x	P	Q	R	S	T	U	V	W
\13x	X	Y	Z	[	\	]	^	_
\14x	`	a	b	c	d	e	f	g
\15x	h	i	j	k	l	m	n	o
\16x	p	q	r	s	t	u	v	w
\17x	x	y	z	{	|	}	~	

Table 22: Symbols for font **-sscp**

	0	1	2	3	4	5	6	7
\00x								
\01x								
\02x								
\03x								
\04x								
\05x								
\06x								
\07x								
\10x		*A*	*B*	*C*	*D*	*E*	*F*	*G*
\11x	*H*	*I*	*J*	*K*	*L*	*M*	*N*	*O*
\12x	*P*	*Q*	*R*	*S*	*T*	*U*	*V*	*W*
\13x	*X*	*Y*	*Z*					
\14x		*a*	*b*	*c*	*d*	*e*	*f*	*g*
\15x	*h*	*i*	*j*	*k*	*l*	*m*	*n*	*o*
\16x	*p*	*q*	*r*	*s*	*t*	*u*	*v*	*w*
\17x	*x*	*y*	*z*					

Table 23: Symbols for font **-sym1**

	0	1	2	3	4	5	6	7
\00x		=	♤	♡	◇	♧	♧	♣
\01x	▲	◓	◣	^	⌒	⌒	◡	⌣
\02x	☈	6	ı	\	∿	⎴	∧	▽
\03x	□	△	◇	☆	+	×	✱	•
\04x	■	▲	◀	▼	▶	★	⚓	⚓
\05x	⚒	Ѧ	≛	⛴	✙	©	✡	🔔
\06x	🌴	🌳	Ϙ	⺌	·	•	∘	○
\07x	○	◯	◯	⛉	⛊	⚑	∞	†
\10x	‡	∃	☉	☿	♀	⊕	♂	♃
\11x	♄	♅	♆	B	☾	☄	✱	☊
\12x	☋	ˈ	ˌ	|	″	ıı	‖	
\13x								
\14x								
\15x								
\16x								
\17x								

Table 24: Symbols for font `-tr`

	0	1	2	3	4	5	6	7
\00x								
\01x								
\02x								
\03x							—	°
\04x		!	"		$		&	'
\05x	(	)	*	+	,	−	.	/
\06x	0	1	2	3	4	5	6	7
\07x	8	9	:	;		=		?
\10x		A	B	C	D	E	F	G
\11x	H	I	J	K	L	M	N	O
\12x	P	Q	R	S	T	U	V	W
\13x	X	Y	Z					
\14x		a	b	c	d	e	f	g
\15x	h	i	j	k	l	m	n	o
\16x	p	q	r	s	t	u	v	w
\17x	x	y	z					

Table 25: Symbols for font `-tri`

	0	1	2	3	4	5	6	7
\00x								
\01x								
\02x								
\03x							—	°
\04x		!	"		$		&	'
\05x	(	)	*	+	,	−	.	/
\06x	*0*	*1*	*2*	*3*	*4*	*5*	*6*	*7*
\07x	*8*	*9*	:	;		‘		?
\10x		*A*	*B*	*C*	*D*	*E*	*F*	*G*
\11x	*H*	*I*	*J*	*K*	*L*	*M*	*N*	*O*
\12x	*P*	*Q*	*R*	*S*	*T*	*U*	*V*	*W*
\13x	*X*	*Y*	*Z*					
\14x		*a*	*b*	*c*	*d*	*e*	*f*	*g*
\15x	*h*	*i*	*j*	*k*	*l*	*m*	*n*	*o*
\16x	*p*	*q*	*r*	*s*	*t*	*u*	*v*	*w*
\17x	*x*	*y*	*z*					

Table 26: Symbols for typesetting font **-nrR**

	0	1	2	3	4	5	6	7
\00x		-	≈	¼	½	¾		¢
\01x	®	©	—	_				
\02x					ff	fi	fl	ffi
\03x	ffl	□	•	°	†			
\04x		!	"	#	$	%	&	’
\05x	(	)	*	+	,	−	.	/
\06x	0	1	2	3	4	5	6	7
\07x	8	9	:	;	<	=	>	?
\10x	@	A	B	C	D	E	F	G
\11x	H	I	J	K	L	M	N	O
\12x	P	Q	R	S	T	U	V	W
\13x	X	Y	Z	[	\	]	^	_
\14x	‘	a	b	c	d	e	f	g
\15x	h	i	j	k	l	m	n	o
\16x	p	q	r	s	t	u	v	w
\17x	x	y	z	{	|	}	~	

Table 27: Symbols for typesetting font **-nrB**

	0	1	2	3	4	5	6	7
\00x		-						
\01x								
\02x								
\03x								
\04x		!	"		$		&	’
\05x	(	)	*	+	,	−	.	/
\06x	0	1	2	3	4	5	6	7
\07x	8	9	:	;		=		?
\10x		A	B	C	D	E	F	G
\11x	H	I	J	K	L	M	N	O
\12x	P	Q	R	S	T	U	V	W
\13x	X	Y	Z					
\14x	‘	a	b	c	d	e	f	g
\15x	h	i	j	k	l	m	n	o
\16x	p	q	r	s	t	u	v	w
\17x	x	y	z					

Table 28: Symbols for typesetting font `-nrI`

	0	1	2	3	4	5	6	7
\00x		–						
\01x				—				
\02x					ff	fi	fl	ffi
\03x	ffl							
\04x		!			$		&	’
\05x	(	)	*		,	-	.	/
\06x	0	1	2	3	4	5	6	7
\07x	8	9	:	;				?
\10x		A	B	C	D	E	F	G
\11x	H	I	J	K	L	M	N	O
\12x	P	Q	R	S	T	U	V	W
\13x	X	Y	Z					
\14x	‘	a	b	c	d	e	f	g
\15x	h	i	j	k	l	m	n	o
\16x	p	q	r	s	t	u	v	w
\17x	x	y	z					

Table 29: Symbols for typesetting font `-nrBI`

	0	1	2	3	4	5	6	7
\00x		–						
\01x								
\02x								
\03x								
\04x		!			$		&	’
\05x	(	)	*	+	,	-	.	/
\06x	0	1	2	3	4	5	6	7
\07x	8	9	:	;		=		?
\10x		A	B	C	D	E	F	G
\11x	H	I	J	K	L	M	N	O
\12x	P	Q	R	S	T	U	V	W
\13x	X	Y	Z					
\14x	‘	a	b	c	d	e	f	g
\15x	h	i	j	k	l	m	n	o
\16x	p	q	r	s	t	u	v	w
\17x	x	y	z					

Table 30: Symbols for typesetting font **-nrS**

	0	1	2	3	4	5	6	7
\00x		_	×	×	÷	≠	≡	≦
\01x	≧	∝		⊂	∪	⊃	∩	∈
\02x	→	↑	←	↓	∂	∇	∫	∞
\03x	§		‡	√	~	±	∓	
\04x	○	⌋		#				
\05x				+		−		
\06x	‾		⎧	⎩	⎫	⎭	⎨	⎬
\07x	│	⎡	⎣	⎤	<	=	>	
\10x	@	A	B	Ξ	Δ	E	Φ	Γ
\11x	Θ	I		K	Λ	M	N	O
\12x	Π	Ψ	P	Σ	T	Υ		Ω
\13x	X	H	Z	[		]		
\14x		α	β	ξ	δ	ε	φ	γ
\15x	ϑ	ι		κ	λ	μ	ν	ο
\16x	π	ψ	ρ	σ	τ	υ	ς	ω
\17x	χ	η	ζ	{	│	}		

Appendix B–UNIX Manual Style Documentation

Because the UNIX manual style documentation must be compatible with UNIX's **nroff**(1) text preparation system, the manual pages for software presented in this report are prepared with the typesetting version **troff** and attached to this appendix. Style and page numbering are necessarily different from the LaTeX typesetting style used in preparing this and previous pages. It is recommended that the following pages be copied and inserted in local user documentation manuals.

In providing supplementary documentation describing the characteristics of individual plotting devices the author decided to place this material in "Chapter" 7 of the UNIX manual system. Only documentation for those devices available to the local system should be added to local manuals.

NAME

plotter - plotting of metagraphic stream

SYNOPSIS

plotter [-diorspxyXYPD [args]] [file(s)]

DESCRIPTION

Plotter interprets a metagraphic stream and reformats the data into a form acceptable by the selected graphics device.

Except for the **–p** and **–x** options the following command line control parameters can appear in any order:

–d *name*

Name defines the name of the plotting device to which the output is to be directed. If *name* is omitted, **plotter** acquires the name of the device from the **GTERM** entry of the processes' environment or the **TERM** entry if **GTERM** is not specified. If *name* or the default terminal is not an implemented device, **plotter** proceeds with a dummy device and does not produce output.

–s *scale*

Scale must be a floating point value which will scale the coordinate values of the input metagraphic stream.

–o *output*

This option directs the device dependent graphic control data to file *output*. If not specified the data will be sent to the output file specified by the GRAPHB environment option or to *stdout* if it is also not specified.

–i *m.n*

This option only applies to the interactive use of **plotter** by programs using a bidirection link. It should not be employed by *shell* execution of **plotter.** *M.n* are automatically generated by graphics library software. *M* is the file descriptor for the input metagraphic data pipe to **plotter** and *n* is the file descriptor of the return data pipe to the calling program.

–q

This option alerts interactive display drivers not to sound console bell and wait for a <CR> at the end of a plot. Its primary purpose is with cursor control applications which hold the screen with cursor input and where plot termination is at user's cursor input control.

–r

If this option is selected, the x and y axis are reversed on the plotting device.

–[**p**|**P**]*m:n[,m:n]*

Mapping the mechanical pens selected in the original overlay files to new mechanical pens may be performed by use of this option. The **–p** option is sensitive to its position on the run line and only affects overlay files that follow its specification. It may be employed on the run-line as often as meaningful.

The argument pair *m:n* (immediately following the **–p** or **–P**) consists of the original mechanical pen number *m* and the new mechanical pen number *n* to be employed in this plot. *M* may also denote a range of original mechanical pen numbers when a – is employed. For example: 3–6:0 maps original values 3 through 6 to new pen 0, 0–3:5 or –3:5 maps pens 0 through 3 to new pen 5, and 3–:2 maps pens 3 through 255 to pen 2.

The **–P** option applies the pen mapping to all input files and it may appear anywhere on the runline.

–[X|Y|x|y]*n*

This option provides for shifting the overlay by *n* overlay coordinate units in the x and/or y axis. The options **–X** and **–Y** define an offset for all overlays while the **–x** and **–y** options create offsets relative to 0,0 or origin modified by **–XY** only for the overlays that follow on the run-line.

Offset units depend upon application software generating the overlay files and the units expected by the plot device. For hardcopy devices this is typically 200 counts/cm so that creating a 10 cm offset in the x axis would require a value for *n* of 10×200 counts or –x2000.

–Dstring

String is information passed to the selected device driver. Reference to specific driver documentation must be made for details of content.

file(s)

The files named contain metagraphic commands compatible with this system (see Device Independent Vector Graphics manual). The files are processed in a left to right order. A – may be used once to designate input from *stdin.* If no files are given and option **–i** is not employed, *stdin* is assumed to be the source of the metagraphic stream.

ENVIRONMENT

Plotter requires the environment entry **GRAPHB** which is also employed by application software employing this graphics system. The general **csh** method of initializing this environment parameter is:

```
setenv GRAPHB prog_path:fonts_path:def_font[:dev,file]:
```

where *prog_path* is the full path name of program **plotter,** *fonts_path* is the directory containing the font definition data, and *def_font* is the default font name in *fonts_path* employed when only a – font name is used. *Dev,file* is an optional list of plotting devices and their output file name. Note that each plotter entry must be separated by a : and associated file delimited with a ,. The following is an example entry:

```
setenv GRAPHB /graph/plotter:/graph/fonts/:sr:kong,/dev/ttym2:
```

EXAMPLE

```
plotter file1 file2 -s 4
```

will generate a composite plot of both files on the user's terminal and will have the coordinates scaled by a factor of 4.

```
plotter file1 -p1:0 file2 -s .25 -d calcomp -o caltemp
```

will scale the meta-graphic files by 1/4 and output the *calcomp* plotter control to the disc file *caltemp.* The mechanical pen 1 of metagraphic *file2* is mapped to pen 0;

SEE ALSO

Device Independent Vector Graphics manual.
Documentation of specific graphic devices.
GRAPHICS(3)
device descriptions(7p)

DIAGNOSTICS

If an invalid graphics device is selected, a message is output to *stderr* and all input data are ignored. Additional error conditions are available only though bidirectional linkage with controlling process.

AUTHOR/MAINTENANCE

Gerald I. Evenden, USGS, Woods Hole, MA 02543

NAME

set970 - set performance factors for Calcomp 970

SYNOPSIS

set970 [n]

DESCRIPTION

Set970 sets the performance factors of the Calcomp 970 plotter. These factors consist of pen acceleration, maximum speed and approximate creep speed. The optional parameter **n** selects from the following table the range of factors selectable. If omitted, a value of 34 is used and if n<0 then 0 and if n>47 then 47.

n	Acc.	Vm.	Vc.	n	Acc.	Vm.	Vc.	n	Acc.	Vm.	Vc.
0	0.6	6	0.5	16	1.0	6	0.5	32	2.0	6	0.5
1	0.6	6	1.0	17	1.0	6	1.0	33	2.0	6	1.0
2	0.6	6	2.0	18	1.0	6	2.0	34	2.0	6	2.0
3	0.6	6	3.0	19	1.0	6	3.0	35	2.0	6	3.0
4	0.6	10	0.5	20	1.0	10	0.5	36	2.0	10	0.5
5	0.6	10	1.0	21	1.0	10	1.0	37	2.0	10	1.0
6	0.6	10	2.0	22	1.0	10	2.0	38	2.0	10	2.0
7	0.6	10	3.0	23	1.0	10	3.0	39	2.0	10	3.0
8	0.6	20	0.5	24	1.0	20	0.5	40	2.0	20	0.5
9	0.6	20	1.0	25	1.0	20	1.0	41	2.0	20	1.0
10	0.6	20	2.0	26	1.0	20	2.0	42	2.0	20	2.0
11	0.6	20	3.0	27	1.0	20	3.0	43	2.0	20	3.0
12	0.6	30	0.5	28	1.0	30	0.5	44	2.0	30	0.5
13	0.6	30	1.0	29	1.0	30	1.0	45	2.0	30	1.0
14	0.6	30	2.0	30	1.0	30	2.0	46	2.0	30	2.0
15	0.6	30	3.0	31	1.0	30	3.0	47	2.0	30	3.0

Output of the control stream is the standard output.

NOTE: the **AUX** switch on the 970 must be enabled for this operation to have any effect.

EXAMPLE

```
( set970 10 ; plotter ... ) > /dev/ttyo4
```

sets performance factor to number 10 prior to execution of **plotter.**

DIAGNOSTICS

None.

AUTHOR/MAINTENANCE

Gerald I. Evenden, USGS, Woods Hole, MA 02543

NAME

symgen - generate symbol tables for program plotter

SYNOPSIS

symgen font_def font_file descriptor_file[s]

DESCRIPTION

Program **symgen** reformats vector information for drafting characters into a binary file structure, *font_file* for use by the graphics driver program **plotter.** The file *font_def* contains a definition of the mapping of the **ascii** character set values and the symbol identifier numbers in the *descriptor_file.*

The first line of the *font_def* file must containing two numeric values: the height of the character set's upper case letters followed by the y axis offset of the character's origin from the character base line. If a non-zero y-offset value is employed, the origin of the characters is shifted from the normal central position to the lower left hand corner. This latter feature is useful when the symbol set is to be used in typesetting applications.

The remaining lines of the *font_def* file contain either an ascii symbol or two digit numeric value of the symbol followed by the symbol number in the *descriptor_file.*

The *descriptor_file* is a compressed version (all unneccessary blanks removed) of the Hershey symbol tables. A line starting with a number is the beginning of a symbol vector definition and the number is the symbol number referenced in the *font_def* file. Entries must be in ascending symbol number order and a line beginning with a : is a continuation of the previous line. The pair of values immediately following the symbol number (subsequent pairs delimited by colons) indicate the extent of the symbol to the left and right of the symbol's center. The remaining symbol pairs are either x–y coordinates drafting the symbol or pen control. Special coordinates 128 0 and 129 0 represent respective pen-up motion to the next value and end of symbol definition (note that the original Hershey usage employed –64 0 and –64 –64 respectively). Pen-up to the first coordinate is always implied.

There may be more than one *descriptor_file* as long as the ascending sequence of the symbol numbers is preserved. This allows local expansion of symbols without modifying the original Hershey definitions. If the *descriptor_file* is omitted or a – is used then stdin is assumed.

Special note: the y coordinates of the original Hershey table are positive downward. The current version of **symgen** reverses this sign convention internally.

EXAMPLE

A sample execution for generating a standard font selection for the program **plotter** is:

```
symgen fonts/basic-fonts/sr.D sr fonts/H????.tab
```

where the partial contents of fonts/basic-fonts/sr.D appear as:

```
21    0     simplex roman, size 21 units
01    899   ascii 0 cannot be used
02    900
...
31    718   degree
32    698   space
!     714
...
0     700   numbers
```

```
1     701
...
A     501     upper case letters
B     502
etc.
```

As demonstrated, comments can follow the second numeric entry.

FILES

H0000.tab standard Hershey symbol set.

H4000.tab expansion symbols referenced on pages 24 and 25 of NBS 424.

By using the Hxxxx.tab nomenclature with the xxxx representing the first symbol number of the file, wild card expansion (i.e. H????.tab) on the runline will ensure proper loading of multiple *descriptor files.*

SEE ALSO

Device Independent Vector Graphics manual.
NBS Special Publication 424, Hershey symbol tables.
Font definition files and Hershey symbol table files in directory fonts.

AUTHOR/MAINTENANCE

Gerald I. Evenden, USGS, Woods Hole, MA 02543

NAME

plotopen, plotend, defopen, defclose, pltflush, pxyxmit, plotopt, plotreq - graphics control

SYNOPSIS

```
#include <graphics.h>

plotopen(argl)
char *argl[3+];

plotend();

defopen(name);
char *name;

defclose();

pltflush();

pxyxmit(cmd, x, y);
int cmd;
long x, y;

plotopt(cmd, [ optval ])
int cmd;
[ ( long | char * ) optval; ]

ANSWR *plotreq(cmd);
int cmd;
```

DESCRIPTION

The information presented here is only a brief synopsis and syntax of entry points of the device independent graphics system.

Plotopen and/or *defopen* are necessary to initialize the graphics system. *Plotopen* establishes a bidirectional link with program **plotter** for direct output of graphics to a particular device. Note that the first and second pointers in the list *argl* are ignored and the last element must be a null pointer (0). When interactively linking to **plotter** one adjacent pair of the arguments must contain '"-i","."' which will indicate the point of pipe linkage with **plotter.** Other arguments may contain other arguments associated with **plotter** execution. If *plotopen* is employed interactively the graphics operations should be terminated by a call to *plotend.*

Defopen establishes an output file which collects all meta-graphics generated by subsequent graphic calls. *Defclose* terminates output of data to the file established by *defopen.*

Note that both *plotend* and *plotclose* will flush current contents of output buffers.

Pxyxmit transmits pen x and y coordinates to the meta-graphic stream. Principally *cmd* is used to designate the status of the "pen" during motion from the last position to the current coordinates. It also may indicate that the current values of x and y are relative to a previous absolute position.

Plotopt entry is used to transmit a wide variety of options, attributes and control to the meta-graphic stream.

If the process has been linked to **plotter** via a *plotopen* call, status, cursor position as well as other information can be retrieved with execution of *plotreq*. The results of the inquiry are returned as elements of the structure pointed to by *plotreq*.

LIBRARY

graphics.h
grerror.h
libgraph.a

SEE ALSO

Device Independent Vector Graphics manual is required reading for a complete description of the graphics options and usage.

PLOTTER(1).

AUTHOR/MAINTENANCE

Gerald I. Evenden, USGS, Woods Hole, MA 02543

NAME

c5800 - Calcomp electrostatic color external device driver for program plotter

SYNOPSIS

plotter -d c5800 ...

DESCRIPTION

This device driver produces output for external driver linking to Calcomp 5800 subroutine library.

The following **–D plotter** command line control parameter can appear in any position on the runline:

–DR*x*,*y*

The values of *x* and *y* are substituted for the respective maximum size of the plotter.

SIZE AND RESOLUTION

Maximum x and y axis sizes are respectively 65535 and 22350 or 327×111cm with a resolution of 200 counts/cm.

SPECIAL STRING OPERATIONS

None.

MECHANICAL PENS

Mechanical pen numbers 0–1023 select Calcomp 5800 pen numbers 1–1024.

NOTE

Requires program **c5800** in same directory as program **plotter.**

SEE ALSO

Device Independent Vector Graphics manual.
Calcomp technical manuals.
plotter(1)

AUTHOR/MAINTENANCE

Gerald I. Evenden, USGS, Woods Hole, MA 02543

NAME

c970 - Calcomp 970 belt-bed plotter

SYNOPSIS

plotter -d c970 ...

DESCRIPTION

The Calcomp 970 is a mechanical pen, belt-bed plotter.

There are no **–D** options associated with this plotter.

SIZE AND RESOLUTION

Maximum x and y axis sizes are respectively 40000 and 26040 (200×130cm) with a resolution of 200 counts/cm.

SPECIAL STRING OPERATIONS

None.

MECHANICAL PENS

Mechanical pens numbers 0–3 select mechanical pens 1–4. Pen types, widths, etc. determined by operator.

NOTES

Some performance characteristics can be controlled by program **set970** before running plot.

SEE ALSO

Device Independent Vector Graphics manual.
set970(1)
plotter(1)

AUTHOR/MAINTENANCE

Gerald I. Evenden, USGS, Woods Hole, MA 02543

NAME

debug - internal debugging driver
exdebug - external debugging driver

SYNOPSIS

plotter -d debug ...
plotter -d exdebug ...

DESCRIPTION

These drivers produce ASCII text interpretation of the drafting information passed to the drivers and are used for testing and debugging applications.

For **exdebug** the following **–D plotter** command line control parameter can appear in any order:

–DR*x,y*

The values of *x* and *y* are substituted for the respective maximum size of the plotter.

Debug accepts and lists any **–D** options.

SIZE AND RESOLUTION

Debug has maximum range of 3000×2000 and **exdebug** has a range of 15000×10000. Resolution not applicable.

NOTE

Exdebug requires program **extdebug** in same directory as program **plotter.**

SEE ALSO

Device Independent Vector Graphics manual.
plotter(1)

AUTHOR/MAINTENANCE

Gerald I. Evenden, USGS, Woods Hole, MA 02543

NAME

gerber - Gerber photo-head plotter

SYNOPSIS

plotter –d gerber ...
plotter –d gerbers ...

DESCRIPTION

The Gerber is a high resolution photohead plotter designed to produce negatives for publication processes. Output should always be to a 9-track, 1600bpi magnetic tape drive. Dependent upon magnetic tape drive, **gerbers** may be required to swab or interchange output byte but this is becoming rare. The output of this driver is also used with the Scitex system.

There are no **–Dx** options for this device.

SIZE AND RESOLUTION

Maximum x and y axis sizes are respectively 24000 and 32000 (120×160cm) with a resolution of 200 counts/cm.

SPECIAL STRING OPERATIONS

None.

MECHANICAL PENS

Mechanical pen no.	Line width (inches)
0	0.002
1	.003
2	.004
3	.005
4	.006
5	.007
6	.008
7	.009
8	.010
9	.012
10	.014
11	.015
12	.025
13	.050

SEE ALSO

Device Independent Vector Graphics manual.
plotter(1)

AUTHOR/MAINTENANCE

Gerald I. Evenden, USGS, Woods Hole, MA 02543

NAME

kong - Kongsberg photo-head plotter

SYNOPSIS

plotter -d kong ...
plotter -d skong ...

DESCRIPTION

The Kongberg is a high resolution photohead plotter designed to produce negatives for publication processes. Output should always be to a 9-track, 1600bpi magnetic tape drive. Dependent upon magnetic tape drive, **skong** may be required to swab or interchange output byte but this is becoming rare.

There are no **-Dx** options for this device.

SIZE AND RESOLUTION

Maximum x and y axis sizes are respectively 24000 and 32000 (120×160cm) with a resolution of 200 counts/cm.

SPECIAL STRING OPERATIONS

None.

MECHANICAL PENS

Spot size in microns	Mechanical pen no.
2	0
3	1
4	2
5	3
6	4
7	5
9	6
10	7
14	8
15	9
21	10
30	11
45	12
60	13
90	14
135	15

SEE ALSO

Device Independent Vector Graphics manual.
plotter(1)

AUTHOR/MAINTENANCE

Gerald I. Evenden, USGS, Woods Hole, MA 02543

NAME

ps - PostScript internal device driver for program plotter

SYNOPSIS

plotter -d ps ...

DESCRIPTION

This device driver produces output compatible with Adobe's PostScript display language.

The following **–D plotter** command line control parameters can appear in any order:

–Dx

Suppress PostScript **showpage** at end of plot. This allows concatenation of output to create composite plot.

–Dt

Suppress PostScript initialization string and **showpage.** This option should be employed when output is employed with programs that convert TeX dvi files to PostScript programs. See note below.

–Do*x,y*

Identical to plotter **–XY** command and may be discontinued.

SIZE AND RESOLUTION

Maximum x and y axis sizes are respectively 5588 and 4318 (11.5×8 inches) with a resolution of 200 counts/cm.

SPECIAL STRING OPERATIONS

None.

MECHANICAL PENS

Eight pen widths may be selected by mechanical pen numbers 0–7 that are respectively: 1, 5, 10, 15, 20, 30, 40 and 50 points (1/72.27 inch) wide. By adding 8, 16 and 24 to the pen number the lines are respectively drafted with **setgray** levels of .25, .5 and .75.

NOTES

The following initialization operations must precede driver output if the **-Dt** option is employed:

```
/L { {rlineto} repeat currentpoint stroke moveto} bind def
 /U { moveto } bind def
 1 setlinecap
 1 setlinejoin
```

SEE ALSO

Device Independent Vector Graphics manual.
plotter(1)

DIAGNOSTICS

Invalid **–D** option will quietly cause initialization failure.

AUTHOR/MAINTENANCE

Gerald I. Evenden, USGS, Woods Hole, MA 02543

NAME

ranger - driver to determine x–y range of metagraphic data

SYNOPSIS

plotter -d ranger ...

DESCRIPTION

The purpose of this driver is to determine the range of x–y coordinates in one or more metagraphic files and print the results as minimum x, maximum x, minimum y and maximum y on the output.

There are no **–D plotter** command line control parameters.

SIZE AND RESOLUTION

Maximum x and y axis sizes are respectively 100000 and 100000 (500m for 200 count/cm resolution).

SEE ALSO

Device Independent Vector Graphics manual.
plotter(1)

AUTHOR/MAINTENANCE

Gerald I. Evenden, USGS, Woods Hole, MA 02543