Fast Alpha/Font Manager Programmer's Manual

HP 9000 Series 300/800 Computers

HP Part Number 98592-90090

HEWLETT PACKARD

Hewlett-Packard Company
3404 East Harmony Road, Fort Collins, Colorado 80525

Notices

MANUAL COMMENT CARD

HP Part Number 98592-90090 E1288

Your comments and suggestions help us determine how well we meet your needs.

Fast Alpha/Font Manager
Programmer's Manual

	Agree				Disagree
The manual is well organized.	○	○	○	○	○
It is easy to find information in the manual.	○	○	○	○	○
The manual explains features well.	○	○	○	○	○
The manual contains enough examples.	○	○	○	○	○
The examples are appropriate for my needs.	○	○	○	○	○
The manual covers enough topics.	○	○	○	○	○
Overall, the manual meets my expectations.	○	○	○	○	○

You have used this product:

__ Less than 1 week	__ Less than 1 year	__ More than 2 years
__ Less than 1 month	__ 1 to 2 years	

fold ——

Please write additional comments, particularly if you disagree with a statement above. Use additional pages if you wish. The more specific your comments, the more useful they are to us.

Comments: ______________________________

____
____
____
____
____
____
____
____
____
____
____

Please Tape Here

Please print or type your name and address.

Name:	
Company:	
Address:	
City, State, Zip:	
Telephone:	

Additional Comments:

Fast Alpha/Font Manager Programmer's Manual
HP Part Number 98592-90090
E1288

Hewlett-Packard Company
Attn: Learning Products Center
3404 East Harmony Road
Fort Collins, Colorado 80525-9988

Printing History

New editions of this manual will incorporate all material updated since the previous edition. Update packages may be issued between editions and contain replacement and additional pages to be merged into the manual by the user. Each updated page will be indicated by a revision date at the bottom of the page. A vertical bar in the margin indicates the changes on each page. Note that pages which are rearranged due to changes on a previous page are not considered revised.

The manual printing date and part number indicate its current edition. The printing date changes when a new edition is printed. (Minor corrections and updates which are incorporated at reprint do not cause the date to change.) The manual part number changes when extensive technical changes are incorporated.

April 1988 ... Edition 1. This manual is valid for HP-UX Release 6.2 on HP 9000 Series 300 Models.

November 1988 ... Update. To add applicability for Series 800 computers and adds font information.

December 1988 ... Edition 2. Replaces edition 1 and all updates.

Contents

1. Overview of Fast Alpha/Font Manager

History of FA/FM 1-1
Outline 1-2
Chapter 1: Overview 1-2
Chapter 2: The Fast Alpha Library 1-2
Chapter 3: The Font Manager Library 1-3
Chapter 4: Compiling, Linking and File Sets 1-3
Reference: The Reference Pages for FA/FM 1-3
Where to Go for Further Information 1-3
Overview of FA/FM Routines 1-3
Fast Alpha Routines 1-3
Font Manager Calls 1-4

2. The Fast Alpha Library

Concepts 2-1
Programming Model 2-2
Get the File Descriptor 2-2
Initialize the Fast Alpha Environment 2-3
Close the Device Interface 2-3
Cursor Positioning 2-3
Character Enhancements 2-4
Fast Alpha Rectangles 2-4
Initializing/Terminating the Fast Alpha Environment . . . 2-6
Initializing the Fast Alpha Environment 2-6
Terminating the Fast Alpha Environment 2-6
Example 2-6
Changing the Fast Alpha Environment 2-8
Getting Environment Information 2-8
Setting Environment Information 2-8

Performance Considerations 2-11
Example 2-12
Cursor Control 2-13
Procedure 2-13
Precautions 2-14
Examples 2-14
Writing Characters 2-15
Writing Character Strings 2-15
Filling a Rectangle 2-16
Example 2-16
Font Manipulation 2-18
Concepts 2-18
Loading a Font 2-19
Activating a Font 2-19
Removing a Font 2-19
Setting Font Colors 2-19
Precautions 2-20
Example 2-20
Clearing a Rectangle 2-21
Example 2-22
Scrolling a Rectangle 2-23
Example 2-23

3. The Font Manager Library
Concepts 3-2
Font Management 3-3
Concepts 3-3
Loading a Font 3-4
Activating a Font 3-5
Removing a Font 3-5
Example 3-6
Writing Characters 3-7
Writing Characters 3-7
Controlling the Write Direction 3-9
Example 3-10
Setting Colors 3-11
Example 3-11
Optimizing Character Generation 3-12

Determining String Length 3-12
Example 3-13
Writing Sixteen-Bit Font Strings 3-13
Character Clipping 3-14
Enabling/Disabling Clipping 3-14
Setting Clip Limits 3-15
Example 3-15
Font Information Routines 3-17
Font Size 3-17
Getting Font Size Information 3-18
Example 3-19
Font Style 3-20
Getting Font Style Information 3-20
Getting a Font's Name 3-21
Example 3-21

4. Compiling FA/FM Programs
Linking Window Libraries 4-1
Examples 4-2
Font and Icon Files 4-2
Header Files 4-3

5. Reference

6. Appendix A

Index

1

Overview of Fast Alpha/Font Manager

The Fast Alpha/Font Manager is two libraries. The fast alpha display library provides high-performance alpha capabilites with bit-mapped displays and graphics windows. The font manager library provides a high-performance, low-level textual interface to bit-mapped displays and graphics windows.

History of FA/FM

The Fast Alpha/Font Manager came from Windows/9000 as a way to write to windows quickly and to merge alpha text (as opposed to graphics text) and Starbase graphics. After the FA/FM libraries were in place, it became desirable to expand them to work with bit mapped displays. This was done, and the libraries remained fairly constant until the Series 800 was developed. Since the Series 800 does not use Windows/9000, FA/FM has been taken out of the Windows/9000 package and packaged with Starbase since FA/FM uses many of the parts of Starbase—drivers, color maps, etc.

There was no functionality change in FA/FM due to the Starbase repackaging.

Note

Starbase graphics was not fully integrated into the initial release of X11. This initial release of X11 is designated by Hewlett-Packard as:

X11 revision A.00.*dd*

The '00' indicates that this is the first release. The *dd* denotes two decimal digits used internally by Hewlett-Packard; these can be ignored. The release of X11 which supports Starbase graphics in an X11 window is designated as:

X11 revision A.01.*dd*

In this document, when X11 is referenced, it refers to the release which supports Starbase in an X11 window (X11 revision A.01.*dd*). When the initial release is discussed, it is referred to specifically as X11 revision A.00.*dd*.

FA/FM has undergone changes to work in the X11 environment. What has changed is:

- FA/FM uses font files in X11 Server Natural Format (SNF). Fonts in the old Windows/9000 format can no longer be used. All Windows/9000 font files are now shipped in SNF as well as the original Windows/9000 format.
- X11 fonts can now be used with the FA/FM libraries. This includes MIT X11 16 bit font files. An additional entry point, `fm_sixteen_bit`, has been added to the FM library to accompany such functionality.

FA/FM works on Series 300/800 with X11 revision A.01.*dd,* Windows/9000, and raw bit mapped displays.

Outline

Chapter 1: Overview

Chapter 2: The Fast Alpha Library

This chapter describes the use of the fast alpha library. The fast alpha library allows you to write alpha text to the display.

Chapter 3: The Font Manager Library

This chapter describes the use of the font manager. In addition to writing to the display, the font manager allows you to select alternate fonts, place the characters exactly on the display, and to alter the display.

Chapter 4: Compiling, Linking and File Sets

This chapter describes the various libraries needed for FA/FM and the order in which they must be linked with the program.

Reference: The Reference Pages for FA/FM

The reference section contains an alphabetic listing and discussion of all the functions in FA/FM. These pages can also be found on-line, using the `man` command.

Where to Go for Further Information

The information contained in this manual refers occasionally to Starbase, Xlib, and Windows/9000. For further information about these subjects, refer to the associated manuals:

- *Starbase Graphics Techniques,* Vols. 1 and 2
- *Windows/9000 Documentation*
- *HP-UX Reference*
- *Starbase Programming with X11*

Overview of FA/FM Routines

Fast Alpha Routines

`fa`	Summary of the fast alpha routines.
`faclear`	Clears a rectangle area of the display.
`facolors`	Sets the font foreground and background colors for the currently activated fast alpha font.
`facursor`	Positions and enables/disables the fast alpha cursor.
`fafontactivate`	Activates a fast alpha font.
`fafontload`	Loads a font file into the font cache and activates it.

`fafontremove`	Removes a font from the font cache.
`fagetinfo`	Gets information about the current fast alpha environment.
`fainit`	Prepares a fast alpha display for output, sets the display's fast alpha environment to default values.
`farectwrite`	Fills a retangular area with a character.
`faroll`	Rolls a rectangular area.
`fasetinfo`	Sets information about the fast alpha environment.
`faterminate`	Terminates the current fast alpha environment.
`fawrite`	Writes characters in the active font.

Font Manager Calls

`fm_activate`	Activates a font.
`fm_clipflag`	Sets clipping flags.
`fm_cliplim`	Sets the clipping limits.
`fm_colors`	Sets the active font's foreground and background colors.
`fm_fileinfo`	Returns the size of cells in a font file.
`fm_fontdir`	Sets the direction to write characters.
`fm_getname`	Translates a font ID to a filename.
`fm_load`	Loads a font into memory.
`fm_opt`	Optimizes character generation if possible.
`fm_rasterinfo`	Returns the size of cells in a font.
`fm_remove`	Removes a font from memory.
`fm_sixteen_bit`	Sets sixteen bit font string flag.
`fm_str_len`	Determines the pixel length of a character string.
`fm_styleinfo`	Returns style information about a font.
`fm_write`	Writes characters to the display.

2

The Fast Alpha Library

The fast alpha display library provides high-performance alpha (textual) capabilities with bit-mapped graphics displays and graphics windows. For example, you can write text and manipulate fonts, you can clear a portion of a display, or you can scroll part of a display. The following topics are covered in this chapter:

- Concepts essential to using fast alpha routines.
- Initializing and terminating the fast alpha environment.
- Changing the fast alpha environment.
- Cursor control.
- Writing characters.
- Font manipulation.
- Clearing part of a display or window.
- Scrolling part of a display or window.

Concepts

This section discusses concepts essential to understanding the use of fast alpha routines. The following topics are discussed:

- The fast alpha programming model.
- Cursor positioning.
- Character enhancements.
- Fast alpha rectangles.

Note Definitions of fast alpha constants and structures are found in the file `/usr/include/fa.h`.

Also, programs that call fast alpha routines require that both the fast alpha and font manager libraries be linked.

Programming Model

You can use fast alpha routines with any bit-mapped display or graphics window. In other words, you can call fast alpha routines to work with bit-mapped displays or windows on bit-mapped displays. It's the same concept as using Starbase graphics routines with either a bit-mapped display or graphics window. (Throughout this chapter unless otherwise indicated, the term display is used to indicate both bit-mapped display and graphics window.)

Note Fast alpha routines are supported on all bit displays but not on all drivers. See the *Starbase Device Drivers Library* manual to determine if fast alpha routines are supported by a particular driver.

Certain tasks must always be performed in programs that call fast alpha routines.

1. Get the file descriptor.
2. Initialize the fast alpha environment.
3. Use the fast alpha environment.
4. Terminate the fast alpha environment.
5. Close the device interface.

Get the File Descriptor

Fast alpha routines require the file descriptor of the display's or the window's opened device interface.

When using fast alpha routines you must obtain a file descriptor for the display by performing a graphics open (`gopen`(3G)) on the display's device interface. The file descriptor returned from the `gopen` function is the one used by fast alpha routines.

Note If you are using fast alpha routines to combine text and graphics, you should obtain separate file descriptors for fast alpha routines and Starbase routines; that is, you must open the display once for fast alpha routines and once for Starbase routines.

Getting separate file descriptors ensures that fast alpha routines work predictably. If you use the same file descriptor for both fast alpha and Starbase routines, the results are unpredictable.

Initialize the Fast Alpha Environment

Before calling any other fast alpha routines, you must **initialize** the fast alpha environment for the display on which the routines operate. Once the fast alpha environment is initialized, you can call fast alpha routines that manipulate the display. When you are finished using the fast alpha routines with a display, the fast alpha environment must be **terminated.** (The section "Initializing/Terminating the Fast Alpha Environment" contains more information on how to do this.)

You can use window library routines on the window before, during, and after initializing and terminating the fast alpha environment; however, you can use fast alpha routines only between initializing and terminating the environment.

Close the Device Interface

The final task that must be performed in fast alpha programs is closing the device interface of the display. Use the `gclose`(3G) on the display's device file.

Cursor Positioning

The position for placing characters is specified by character column and line, rather than display pixels. The leftmost column of the display is column 0; the topmost row is row 0.

The pixel coordinate equivalents of column-row depend on the size of the current font—the smaller the font, the smaller the pixel coordinates; the larger the font, the larger the pixel coordinates. The results are unpredictable when you use proportionally spaced fonts because fast alpha routines use the height and width of the largest character for determining spacing.

Character Enhancements

Each character may be enhanced with one or more video enhancements. Inverse video and underlining are currently the only enhancements supported.

When you initialize the fast alpha environment, font colors default to white foreground and black background—white characters on a black background. Through fast alpha routines, you can redefine the font foreground and background colors.

Fast Alpha Rectangles

Many fast alpha routines reference rectangles. Rectangles are your means of specifying a particular subset of the display area (in columns and lines) for a fast alpha operation involving more than one line. The rectangle structure is defined in `/usr/include/fa.h` as:

```
struct fa_coordinate
{
 int x, y;
};

struct fa_rectangle
{
 struct fa_coordinate origin; /* included in displayed rectangle */
 struct fa_coordinate corner; /* not included in rectangle    */
};
```

As the comment indicates, the lower-right-corner character is not included in the rectangle as is consistent with C-language arrays. For example, fa_rect defined as:

```
struct fa_rectangle fa_rect;     /* fast alpha rectangle structure */

fa_rect.origin.x = 2;
fa_rect.origin.y = 1;
fa_rect.corner.x = 4;
fa_rect.corner.y = 4;
```

specifies the following rectangle:

```
                columns

                 012345
                0.... ..
                1..XX..
rows            2..XX..
                3..XX..
                4.... ..
                5.... ..
```

This method of display access is fairly low-level, and you may want to build a "friendlier" interface upon this base. The main purpose of the fast alpha routines is to provide you with a fast and intuitive method for getting alpha information on the display.

Initializing/Terminating the Fast Alpha Environment

The `fainit` routine initializes the fast alpha environment for a display device; the `faterminate` terminates a display's fast alpha environment—that is, it releases resources allocated when `fainit` was called.

Initializing the Fast Alpha Environment

To initialize a display's environment, call `fainit`.

> `fainit`*(gfd, driver)*

Set the *driver* parameter to `FAWINDOW`. The *gfd* parameter is the file descriptor for the display.

Initializing the fast alpha environment causes environment information to be allocated for the display. This information affects how fast alpha routines work with the display. You can inquire and change this information with fast alpha routines (discussed in the next section, "Changing the Fast Alpha Environment").

Terminating the Fast Alpha Environment

To terminate a display's environment, call `faterminate`.

> `faterminate`*(gfd)*

Calling this routine deallocates fast alpha environment information for the display device represented by *gfd*. (In order to use fast alpha routines again, you must call `fainit` again.)

Example

When compiling a fast alpha program, you must compile it with several libraries. These libraries are described fully in the chapter "Compiling FA/FM Programs". The following shell script is used to compile most of the example programs in this manual. Remember to use the device driver for your display interface (98550 is used in these examples).

```
PROG=$1
cc -o $PROG $PROG.c -lfa -lfontm -ldd98550 -lsb1 -lsb2
```

The following code segment exemplifies the structure of programs that call fast alpha routines.

```
#include <fa.h>                           /* get fast alpha defs        */
#include <starbase.c.h>                   /* get starbase defs          */

main()     /* program "Structure.c"      */
{
    int gfd;                        /* file descriptors          */

                                          /* open the device or window  */
    if ((gfd = gopen("/dev/crt", OUTDEV, "hp98550", INIT)) == -1)
        exit(1);

    fainit(gfd, FAWINDOW);            /* initialize the fast alpha env. */
/************************************************************************/
/*     Do writes and other things here                                  */
/*                                                                      */
/************************************************************************/

    faterminate(gfd);                     /* terminate the environment  */
    gclose(gfd);                          /* close the device           */
}
```

Changing the Fast Alpha Environment

As described in the previous section, fast alpha environment information is allocated when you initialize the environment. This information affects the manner in which fast alpha routines work.

The exact information maintained in the fast alpha environment is defined by the *fainfo* structure in the header file *fa.h*. The table on the following page briefly describes each of *fainfo*'s fields; for more information on this structure and its values, see *fa.h* and the reference pages for *fasetinfo*.

Getting Environment Information

To get the current fast alpha environment for a display, call **`fagetinfo`**.

`fagetinfo`*(gfd, fainfoptr)*

The *fainfoptr* parameter is a pointer to a *fainfo* structure as defined in *fa.h*. After calling `fagetinfo`, the fields of the structure return the fast alpha environment values specified by *gfd*.

Setting Environment Information

To set fast alpha environment parameters for a display, call **`fasetinfo`**.

`fasetinfo`*(gfd, fainfoptr)*

The *fainfoptr* parameter is a pointer to a *fainfo* structure containing the new values for the environment. You can set only the following parameters with this routine:

- `defaultenhancements`
- `clearbeforewrite`
- `colormode`
- `makecurrent`

Table 2–1. The fainfo Structure[1]

Field	Description	Range
`size`	This is a rectangle structure as defined in the "Concepts" section; it defines the display size.	Limits of display
`capabilities`	You may use the contents of this field to detect what additional capabilities are available on a particular device.	`FAWINDOW`
`enhancements`	This bit-mask defines the default enhancements that are supported on the display device.	(see `fa.h`)
`defaultenhancements`[2]	Initially set to a value that optimizes the performance of the display (`FAOFF`).	(see `fa.h`)
`cursor`	`TRUE` if the cursor can be physically removed from the display device and is `FALSE` otherwise. Currently always `TRUE`.	`TRUE/FALSE`
`fontcellheight` `fontcellwidth`	Indicate the pixel height and width of the active font.	
`clearbeforewrite`[2]	Determines whether the background is automatically cleared before writing characters. The default value is `TRUE`, which causes the background to clear before writing.	`TRUE/FALSE`

1 All fields of the *fainfo* structure are 32-bit integers; this provides compatibility with other languages.

2 You can change only these fields with *fasetinfo*.

The fainfo Structure (continued)

Field	Description	Range
`foregroundplanes` `backgroundplanes`	Specifies the number of memory planes available for controlling the foreground and background colors, respectively.	0, 1, 4, or 8
`colormode`[2]	Indicates color option currently in use. Default is `FAWONB` (white on black).	`FAWONB`, `FACOLOR`, `FABONW`
`makecurrent`[2]	This bit-mask controls the updating of fast alpha operations to the display. Various bits in the mask control when information is displayed with fast alpha routines.	(see `fa.h`)

[2] You can change only these fields with *fasetinfo*

Performance Considerations

- You can set `defaultenhancements` to a different value, however, it initially contains the value that makes the fast alpha library work most efficiently (that is, `FAOFF`). Therefore, changing the value of this parameter may degrade system performance.
- The default value for `clearbeforewrite` is `TRUE` which clears the display background before writing any characters. This ensures that the space where characters are displayed is properly cleared, so that new characters are readable. You can change this value to `FALSE`, in which case the background is not cleared before writing, and you are responsible for controlling the background area.
- Changing the `colormode` parameter to `FACOLOR` causes the fast alpha routines to run slower. Black and white colors (the default) cause the routines to run faster.
- To increase the speed of fast alpha routines, you may wish to suppress the updating of the display until several write operations are queued. Then, when updating is desired, signal the fast alpha environment to update by setting the `makecurrent` field to `MCALWAYS`. Queued operations are displayed at that time. Then reset `makecurrent` so that operations are queued up—that is, set the bits in `makecurrent` that suppress display updates (see *fa.h*). By doing this you are making effective use of the Starbase buffering facility.

  The default value is `MCALWAYS`, which updates the display after every fast alpha call and may degrade system performance (compared to queueing).

Example

The following code segment sets `clearbeforewrite` to TRUE and sets `makecurrent` so that the display won't be updated for any *fawrite* operations—that is, writes are queued. Later on, `makecurrent` is reset so that all queued *fawrite* operations are performed.

```
#include <fa.h>                                 /* get fast alpha defs       */
#include <starbase.c.h>                         /* get starbase defs         */

main()     /* program "Fainfo.c"      */
{
    int gfd;                              /* file descriptors           */
    struct fainfo fa_env;                       /* FA environment structure  */
                                                /* open the device or window */
    if ((gfd = gopen("/dev/crt", OUTDEV, "hp98550", INIT)) == -1)
        exit(1);

    fainit(gfd, FAWINDOW);                  /* initialize the fast alpha env. */

    if (fagetinfo(gfd, &fa_env) < 0) {       /* get the current environment */
       perror("fagetinfo gfd");
       exit(1);
       }

/*     Set the appropriate values in the structure and call fasetinfo:
       Always clear the area before writing
       Do not update area until the makecurrent field is reset        */

    fa_env.clearbeforewrite = TRUE;
    fa_env.makecurrent = (NOMCONFAWRITE | NOMCONFARECTWRITE);
    if (fasetinfo(gfd, &fa_env) < 0) {
       perror("fasetinfo gfd NOMCONFAWRITE(S)");
       exit(1);
       }
/**************************************************************************/
/*     Do writes and other things here                                   */
/*                                                                       */
/**************************************************************************/

/*  Set makecurrent to make current always to dispaly queued writes     */
    fa_env.makecurrent = MCALWAYS;
    if (fasetinfo(gfd, &fa_env) < 0) {
       perror("fasetinfo gfd MCALWAYS");
       exit(1);
       }

    faterminate(gfd);                    /* terminate the environment  */
    gclose(gfd);                         /* close the device           */
}
```

Cursor Control

With fast alpha routines, you can display and move a cursor on displays. The `facursor` routine performs cursor control operations.

Procedure

To move and/or turn the cursor on or off, call `facursor`.

> `facursor`*(gfd, column, line, cflag)*

The *column* and *line* parameters specify the column and line at which to position the cursor; the top line of the display is line 0, and the leftmost column is column 0. If either *column* or *line* is invalid (or equals `FACURSORNOMOVE`), the cursor's position is not updated.

The *cflag* parameter determines whether or not the cursor is displayed. If *cflag* is `TRUE`, the cursor is displayed; if `FALSE`, the cursor is turned off. Turning the cursor on or off doesn't in itself change the cursor position as fast alpha remembers it.

If you specify invalid coordinates for *column* and *line*—specifically, `FACURSORNOMOVE` as defined in *fa.h*—the cursor won't move, but `cflag` is still effective. This is useful if you wish to turn on/off the cursor at its current position.

If part of the window is off screen, the desired cursor position may also be off screen. For example, if the upper-left corner of the window is off screen, 0, 0 are valid cursor coordinates, but the cursor is not be visible; it is off screen.

Also, you can specify a cursor position that might be occluded by windows higher up in the display stack.

Precautions

The cursor is actually a displayable character taken from the currently active font. Therefore, if no font is activated, the cursor is automatically turned off. Attempting to turn the cursor on when no font is active results in an error.

Examples

The following program turns on the cursor at column 27 and line 12. It pauses for 3 seconds and then turns off the cursor.

```
#include <fa.h>                         /* get fast alpha defs        */
#include <starbase.c.h>                 /* get starbase defs          */

main()     /* program "Cursor.c"          */
{
    int gfd;                      /* file descriptors           */

                                        /* open the device or window  */
    if ((gfd = gopen("/dev/crt", OUTDEV, "hp98550", INIT)) == -1)
        exit(1);

    fainit(gfd, FAWINDOW);          /* initialize the fast alpha env. */

    facursor(gfd, 27,12, TRUE);
    sleep(3);
    facursor(gfd, FACURSORNOMOVE, FACURSORNOMOVE, FALSE);

    faterminate(gfd);                   /* terminate the environment  */
    gclose(gfd);                        /* close the device           */
}
```

Writing Characters

Fast alpha routines provide two kinds of writing operations. You can write a string of characters, or you can fill a rectangle with a specific character. Following are separate discussions for writing strings and filling rectangles.

Writing Character Strings

To write character strings, use the `fawrite` routine.

`fawrite`*(gfd, column, line, charbuf, ebuf, nchars)*

The *column* and *line* parameters specify the character location where the string should start in the display.

The *charbuf* parameter is a pointer to the buffer of characters to be written. This buffer contains *nchars* characters. That is, `fawrite` writes *nchars* characters, taking characters from the address specified by *charbuf*.

The *ebuf* parameter is a pointer to a buffer of enhancements to be applied to each character in *charbuf; ebuf* can be either `NULL` (no characters at all) or can contain *nchars* characters:

- If *ebuf* is `NULL`, then the enhancements specified in `defaultenhancements` are made to each character in the output string.
- Otherwise, each character in *ebuf* defines the enhancement(s) (such as inverse or underlining) to use when displaying the corresponding character in *charbuf*. For example, the fifth character in *ebuf* defines the enhancement(s) to use when displaying the fifth character in *charbuf*.

Valid enhancements are:

`FAOFF`	`@`
`FAINVERSE`	`B`
`FAUNDERLINE`	`D`

You can use the full constant name or the letter.

Filling a Rectangle

The `farectwrite` routine fills a rectangular area in the display that is specified by a rectangle structure.

`farectwrite`*(gfd, character, enhancement, rp)*

The *rp* parameter is a pointer to a rectangle structure that defines the area to be filled. The area is filled with the character specified by the *character* parameter, and the *enhancement* parameter describes the enhancement(s) to use when displaying the character. (If *enhancement* is `NULL`, then `defaultenhancements` are used when displaying *character*.)

Example

The following code segment fills a display rectangle with inverse video X's; the rectangle's upper-left corner is at the origin (column 0, line 0). It then writes the message:

```
 What an exciting
rectangle this is!
```

and underlines the word exciting.

```
#include <fa.h>                              /* get fast alpha defs         */
#include <starbase.c.h>                      /* get starbase defs           */

main()     /* program "Structure.c"       */
{
    int gfd;                           /* file descriptor           */
    struct fa_rectangle rp;                  /* rectangle structure         */
```

```
                                                /* open the device or window  */
    if ((gfd = gopen("/dev/crt", OUTDEV, "hp98550", INIT)) == -1)
        exit(1);

    fainit(gfd, FAWINDOW);          /* initialize the fast alpha env. */

    rp.origin.x = 2;              /* define the area of the rectangle */
    rp.origin.y = 5;
    rp.corner.x = 23;
    rp.corner.y = 9;
    farectwrite(gfd, 'X', FAINVERSE, &rp);  /* write Xs into the rectangle */

/* Now  write the first line into the rectangle:
           @ --> FAOFF (no enhancements)
           D --> FAUNDERLINE (underline the text)
    B --> FAINVERSE (inverse the text)                         */
    fawrite(gfd, 4, 6, "What an exciting", "@@@@@@@@DDDDDDDD", 16);
    fawrite(gfd, 3, 7, "rectangle this is!", NULL, 18);  /* no enhancements */

    faterminate(gfd);                   /* terminate the environment  */
    gclose(gfd);                        /* close the device           */
}
```

Font Manipulation

The fast alpha library contains font manipulation routines you can use to display different fonts. See Appendix A for information about specific font descriptions.

Concepts

At fast alpha initialization time, a default font is established. If a font has already been established with font manager routines (discussed in the next chapter), that font is used. If there is no active font at initialization, a system default font is activated (the font specified by the WMBASEFONT environment variable). You can change the current font with fast alpha routines or font manager routines; however, it is recommended that once you've started using the fast alpha environment, you should make font changes using only the fast alpha routines. This ensures that the fast alpha environment is always aware of the current font attributes (such as height, width, and colors).

Unlike the term0 font management model, there is no notion of base and alternate fonts, there is only the **active** font. Any text written is always displayed in the active font.

Fonts are loaded into the fast alpha font cache from the font directories described in the term0 font management model. Loading a font causes it to be the active font. The fast alpha font cache is not the same one used by term0 font management routines. However, the fast alpha font cache is the same as the font manager's. (In fact, to perform font management, the fast alpha routines call font manager routines.)

When you are through using a font, you can remove it from the font cache.

Fast alpha fonts are often denoted by font ids. These are *not* the same as term0 font ids but are the same as font manager font ids. Attempting to intermix term0 and fast alpha font ids may result in unpredictable system behavior.

Loading a Font

To load a font into your font cache, call the `fafontload` routine.

`fafontload`*(gfd, path)*

The *path* parameter is the path name of the font file to load. When the font is loaded, it is automatically activated, and `fafontload` returns a font id that identifies the font. This font id is required as a parameter to some other font routines. This font id is local to the associated *gfd*. It is not a valid id for any other *gfd*, nor is it valid to any other process.

Activating a Font

To activate a loaded font that isn't currently active, call `fafontactivate`.

`fafontactivate`*(gfd, fontid)*

This routine activates, as the current font, the font specified by *fontid* (*fontid* is the value returned by `fafontload`). After calling this routine, any text written subsequently is displayed in the new font.

Removing a Font

When you are finished using a font, you can remove it from the font cache. The `fafontremove` routine removes a font from the cache.

`fafontremove`*(gfd, fontid)*

After calling this routine, the font specified by *fontid* no longer exist in the font cache. To use this font again, you must reload and reactivate it.

Setting Font Colors

You set the foreground and background colors used when displaying fonts with the `facolors` routine.

`facolors`*(gfd, foreground, background)*

After calling this routine, the active font's foreground and background colors are set to those specified by the *foreground* and *background* parameters. These colors are indices into the Starbase color map.

Supported values are determined by the display device: 0 or 1 for monochromatic displays, 0 to 15 for 4-plane color, and 0 to 255 for 8-plane color.

If the `colormode` field of the *fainfo* structure is not set to `FACOLOR`, the system ignores any color changes.

Calling this routine also causes the fast alpha environment to take note of the current font attributes (that is, width, height, color, etc.).

Precautions

The cursor is actually a displayable character and is taken from the currently active font. Therefore, if no font is activated, the cursor is automatically turned off. Attempting to turn the cursor on when no font is active results in an error.

Example

The following code segment loads an 8 by 16-pixel bold font into the font cache, activates the font, then writes "HELLO" several times in various colors.

```
#include <fa.h>                               /* get fast alpha defs        */
#include <starbase.c.h>                       /* get starbase defs          */

main()      /* program "Fonts.c"               */
{
    int gfd;                            /* file descriptor           */
    int fid;                                  /* font id                    */
    int i;                                    /* index for looping          */
    struct fainfo fa_env;                     /* FA environment structure   */

    if ((gfd = gopen("/dev/crt", OUTDEV, "hp98550", INIT)) == -1)
        exit(1);

    fainit(gfd, FAWINDOW);              /* initialize the fast alpha env. */

    fagetinfo(gfd, &fa_env);               /* get the current environment */
    fa_env.colormode = FACOLOR;            /* set up the color mode       */
    fasetinfo(gfd, &fa_env);

                                 /* Load the bold font into the font cache */
    if ((fid = fafontload(gfd,
        "/usr/lib/raster/8x16/SNF/lp.b.8U")) == -1) {
        perror("fafontload gfd");
```

```
          exit(1);
          }
                                        /* Activate the font           */
         if (fafontactivate(gfd, fid) == -1) {
                 perror("fafontactivate gfd");
                 exit(1);
         }

       for(i=0; i<8; i=i+2){         /* Change colors and write message */
   facolors(gfd, i, i+1);
     fawrite(gfd, i,i,"HELLO",NULL,5);
       }

                                  /* Remove the bold font from the cache */
         if (fafontremove(gfd, fid) == -1) {
            perror("fafontremove gfd");
            exit(1);
            }

      faterminate(gfd);                    /* terminate the environment  */
      gclose(gfd);                         /* close the device           */
  }
```

Clearing a Rectangle

You can clear or erase any rectangular area of characters in a display. For example, you could clear the entire display. The `faclear` routine is used for this purpose.

To clear a rectangle, call the `faclear` routine.

`faclear`*(gfd, enhancements, rp)*

The *rp* parameter is a pointer to a rectangle structure that defines the rectangle to clear.

The *enhancements* parameter is currently ignored by the system and is reserved for future expansion. Leave this parameter set to `FAOFF`.

Example

The following program fills a rectangle with Xs. It then calls a subroutine to clear a different rectangle.

```
#include <fa.h>                             /* get fast alpha defs        */
#include <starbase.c.h>                     /* get starbase defs          */

main()     /* program "Clear.c"            */
{
    int gfd;                            /* file descriptor          */
    int fid;                                /* font id                    */
    int i;                                  /* index for looping          */
    struct fa_rectangle rp;                 /* rectangle structure        */

    if ((gfd = gopen("/dev/crt", OUTDEV, "hp98550", INIT)) == -1)
        exit(1);

    fainit(gfd, FAWINDOW);            /* initialize the fast alpha env. */

    rp.origin.x = 0;
    rp.origin.y = 0;
    rp.corner.x = 15;
    rp.corner.y = 10;
    farectwrite(gfd, 'X', NULL, &rp);

    clear_gr(gfd, 5, 5, 10, 10);

    faterminate(gfd);                       /* terminate the environment  */
    gclose(gfd);                            /* close the device           */

} /* end of main  */

/*************************************************************************/

clear_gr(gfd, row, col, x_chars, y_chars)

    int gfd;                      /* gopened file descriptor           */
    int row, col;                 /* starting row and column           */
    int x_chars, y_chars;         /* number of characters              */
{
    struct fa_rectangle rect;     /* rectangle to be cleared           */

       rect.origin.x = row;       /* define the rectangle to be cleared */
       rect.origin.y = col;
```

```
        rect.corner.x = row + x_chars;
        rect.corner.y = col + y_chars;
                                                /* clear the rectangle   */
        if (faclear(gfd, FAOFF, &rect) == -1) return(-1);
        return(0);

    } /* end of clear_gr */
```

Scrolling a Rectangle

You can scroll any display area defined by a rectangle structure. The `faroll` routine performs this task.

> `faroll`*(gfd, how, howfar, rp)*

The *rp* parameter points to a rectangle structure that defines the portion of the display to scroll. The *how* parameter defines the direction to scroll, and the *howfar* parameter defines how many character units to scroll in the direction indicated by *how*.

The following are valid values for *how:*

- `FAROLLUP` (`'u'`)—roll the rectangle's contents up.
- `FAROLLDOWN` (`'d'`)—roll the rectangle's contents down.
- `FAROLLLEFT` (`'l'`)—roll the rectangle's contents left.
- `FAROLLRIGHT` (`'r'`)—roll the rectangle's contents to the right.

If you move text out of the rectangle area using scrolling, and then move it back into the area, the text is not rewritten.

Example

The following code segment rolls a display's contents in all four directions: up, right, down, left.

```
    #include <fa.h>                         /* get fast alpha defs       */
    #include <starbase.c.h>                 /* get starbase defs         */

    main()     /* program "Roll.c"            */
```

```
{
    int gfd;                         /* file descriptor          */
    int fid;                               /* font id                  */
    int i;                                 /* index for looping        */
    struct fa_rectangle rp;                /* rectangle structure      */

    if ((gfd = gopen("/dev/crt", OUTDEV, "hp98550", INIT)) == -1)
        exit(1);

    fainit(gfd, FAWINDOW);             /* initialize the fast alpha env. */

    rp.origin.x = 4;                   /* outline the rectangle          */
    rp.origin.y = 4;
    rp.corner.x = 16;
    rp.corner.y = 11;
    farectwrite(gfd, 'X', NULL, &rp);

    rp.origin.x = 5;                /* make the rectangle 1 unit smaller */
    rp.origin.y = 5;
    rp.corner.x = 15;
    rp.corner.y = 10;
    farectwrite(gfd, 'Y', FAINVERSE, &rp);  /* this data will move    */
    sleep(1);

/* Roll the contents of the window up 1 line                         */
       if (faroll(gfd, 'u', 1, &rp) == -1) {
           perror("faroll up");
           exit(1);
       }
       sleep(1);
/* Roll the contents of the window right 1 characters                */

       if (faroll(gfd, 'r', 1, &rp) == -1) {
           perror("faroll right");
           exit(1);
       }
       sleep(1);
/* Roll the contents of the window down 2 line                       */

       if (faroll(gfd, 'd', 2, &rp) == -1) {
           perror("faroll down");
           exit(1);
       }
       sleep(1);
/* Roll the contents of the window left 3 characters                    */
       if (faroll(gfd, 'l', 3, &rp) == -1) {
           perror("faroll left");
           exit(1);
       }

    faterminate(gfd);                      /* terminate the environment  */
    gclose(gfd);                           /* close the device           */

}
```

3

The Font Manager Library

The font manager library provides a high-performance, low-level textual interface to graphics windows and bit-mapped displays. This library's functionality overlaps with the fast alpha library, and in fact, some fast alpha routines call font manager routines. However, the font manager provides some powerful capabilities not provided by the fast alpha library.

There are two main differences between the font manager and fast alpha libraries:

1. Font manager uses pixel units to specify character coordinates; fast alpha uses character column and line addressing.
2. Font manager can operate with proportionally spaced fonts; fast alpha cannot.

The following topics are discussed in this chapter:

- Concepts essential to using font management routines.
- Font management.
- Writing characters.
- Character clipping.
- Font information routines.

Concepts

Font manager routines can be used with either bit-mapped displays or graphics windows. The routines require the file descriptor returned from performing a graphics open (`gopen`(3G)) on the device interface for the display.

Note Fast alpha routines are supported on all bit displays but not on all drivers. See the *Starbase Device Drivers Library* manual to determine if fast alpha routines are supported by a particular driver.

The font manager is a distributed library that is controlled by the graphics resource manager daemon (grmd) and a set data structure kept in shared memory common to all users of the font manager. This holds true in all environments; X11 revision A.01.*dd*, Windows/9000, and raw bit mapped displays.

Font manager routines allow you to load, activate and remove fonts, and change attributes that affect how a font is displayed. The font management model is identical to that used by fast alpha routines. In fact, fast alpha routines call font manager routines to perform font management tasks.

Fonts on the system will never be loaded more than once from the same font file. Once a font is loaded by a process, any other process can open and share a copy of that font. The shared copy will not be removed from the system until all processes have released the font.

Note Definitions from the `/usr/include/fonticon.h` header file are used throughout this chapter.

Font Management

The font manager library contains font management routines used to display different fonts on displays.

Concepts

Like the fast alpha routines, the font manager maintains a font cache (or font table). The font table is an area of memory used by the font manager. Font information is loaded into the font table from font files.

Windows/9000 font files are stored in the directory—subdirectory `/usr/lib/raster`. See Appendix A for information about specific Windows/9000 font descriptions.

Fonts under `/usr/lib/raster` are in two formats—the original Windows/9000 format and Server Natural Format (SNF). The SNF versions are used by the FA/FM libraries with the Series 300 6.5 release, Series 800 3.1 release, or later releases. The SNF versions are always located in a subdirectory below the original Windows/9000 version. The subdirectory is called `SNF`. For example, there is a Windows/9000 format font `/usr/lib/raster/8x16/lp.8U`; its corresponding SNF font is `/usr/lib/raster/8x16/SNF/lp.8U.scf`.

The Series 300 6.5 release, Series 800 3.1 release, or later releases of the FA/FM libraries can also load X11 fonts. The X11 fonts are also in SNF and can be found under the directory `/usr/lib/X11/fonts`.

When you load a font with font manager routines, it automatically becomes the active font. Text is always displayed in the active font. You use font manager routines to activate any font you have loaded.

In addition, when you load a font, a unique font id is returned. This font id is used to identify the font to certain font management routines. Font manager font ids are different than those used by term0 font management routines; these ids should not be mixed.

Loaded fonts can be different sizes. By using font manager routines, you can mix different-sized fonts on the same display.

When you are finished using a font, you must always remove it from the font table. Loading and removing a font is analogous to opening and closing a file—after you open a file, you must eventually close it.

As mentioned previously, with windows the font table is shared by all users of the font manager library. Therefore, several users may be using the same font in the font table. The font manager takes care not to duplicate fonts in the font table. When a user attempts to load an already-loaded font, the font manager takes note that another user is using the font. It does not reload the font into the table.

The same is true for removing fonts. If more than one user is using the same font, the font manager doesn't actually remove the font from the table; it takes note that one less user is using the font. If only one user is using a font, removing the font causes the font manager to remove it from the table.

Never assume that a font exists in the font table unless you've loaded it and haven't yet removed it. If you remove a font, there's no guarantee that it still exists in the table because others using the font might remove it. Remember: Don't make any assumptions about the shared memory. Load and remove fonts as if you're the only user.

The foreground and background colors of the active font default to white on black. On color systems, you can redefine the foreground and background colors to any from the system color map.

Loading a Font

To load a font into the font table, call `fm_load`.

`fm_load`*(gfd, path, fontid)*

The path parameter points to the path name of the font file to load into the font table. The *fontid* parameter returns the font's id.

The Series 300 6.5 release, Series 800 3.1 release, or later releases of the FA/FM libraries will only load SNF font files. If the font to be loaded is not an SNF version, the FA/FM libraries will automatically try to load a font from the subdirectory `SNF`. Fonts in standard Server Natural Format are suffixed by `.snf`. The FA/FM libraries also know how to load compressed Server Natural Fonts. These fonts are suffixed by `.scf`.

If the Series 300 6.5 release, Series 800 3.1 release, or later releases of the FA/FM libraries were told to load the font `/usr/lib/raster/8x16/lp.8U`, they would try to open one of the following files:

```
/usr/lib/raster/8x16/lp.8U.snf
/usr/lib/raster/8x16/lp.8U.scf
/usr/lib/raster/8x16/SNF/lp.8U.snf
/usr/lib/raster/8x16/SNF/lp.8U.scf
```

In this order, the first file that was a valid SNF file would be loaded. Thus, programs that used FA/FM libraries released before the Series 300 6.5 release, Series 800 3.1 release, or later releases will not need to have their font paths changed.

In addition to being loaded, the font automatically becomes the active font. So if you want a font other than the one loaded to be the active font, you must activate the other font.

When you are finished using a loaded font, you must remove it.

Activating a Font

To activate a previously loaded font, call `fm_activate`.

`fm_activate`*(gfd, fontid)*

The font represented by the *fontid* parameter is made the active font. All text is displayed in the new font until the next call to `fm_activate`, or until you load a new font.

When you activate a new font, the color is re-set to black and white.

Removing a Font

To remove a loaded font, call `fm_remove`.

`fm_remove`*(gfd, fontid)*

After calling this routine, the font specified by *fontid* is removed from the font table (as far as *your* application is concerned).

If you've accidentally loaded a font twice, you must also remove it twice.

Example

The structure of a font manager program is shown in the following example.

```
#include <starbase.c.h>                  /* get starbase defs          */

main()     /* program "FM_Struct.c"      */
{
    int gfd;                       /* file descriptor            */
    int fid;                             /* font id                    */

    if ((gfd = gopen("/dev/crt", OUTDEV, "hp98550", INIT)) == -1)
        exit(1);

                /* Load the font; the font is automatically activated */
    fm_load(gfd, "/usr/lib/raster/8x16/SNF/lp.8U.scf", &fid);

/***********************************************************************
*                                                                      *
*     Do writes and other routines here                                *
*                                                                      *
***********************************************************************/

    fm_remove(gfd, fid);                 /* Remove the font            */
    gclose(gfd);                         /* Close the device           */
}
```

Writing Characters

With font manager routines, you can write text in the active font to any display. By default, characters are written from left to right; however, characters can be written in any direction—up, down, to the right, or to the left. You can write the characters in any color supported on your system. In addition, you can optimize the generation of characters on your particular display hardware.

Writing Characters

The `fm_write` routine displays character strings on a display.

`fm_write`*(gfd, x,y, str, numchars, dump, colormode)*

The *str* parameter points to the string to write, and *numchars* indicates the number of characters in the string.

The pixel coordinantes indicating where to write the string are specified by *x,y*. The upper-leftmost pixel in the contents area has coordinates *0,0*.

Current write direction determines how the characters are positioned with respect to the *x,y* coordinates. The next figure illustrates character positioning.

The *dump* parameter indicates whether or not to immediately update the display after the write: `TRUE` means to update; `FALSE` means to let the system-imposed buffering take care of the visual update.

The *colormode* parameter determines whether or not to use colors from the previous call to `fm_colors`. If `TRUE`, the area where the characters are written is cleared to the current background color, and the characters are written in the foreground color. Setting this parameter to `TRUE` is analogous to setting `clearbeforewrite` to `TRUE` in the fast alpha environment. This mode has the side effect of leaving the *colormode* and write-enable masks set as needed.

If *colormode* is `FALSE`, the characters are displayed using the current Starbase graphics replacement rule and write mask. For example, if the current replacement rule is to *OR* the image onto the background, the characters are placed over the background image without erasing it. Setting *colormode* to `FALSE` is somewhat analogous to setting `clearbeforewrite` to `FALSE` in the fast alpha environment.

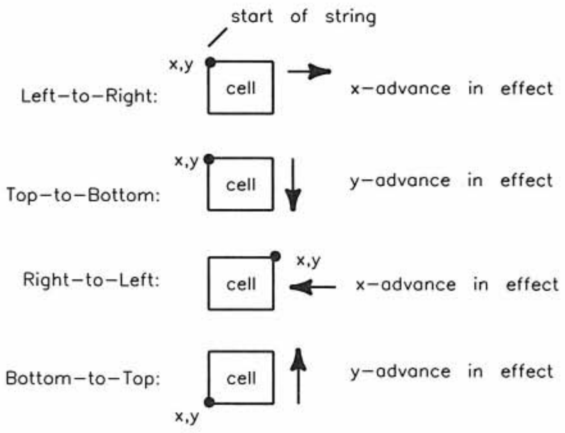

Figure 3–1. Character Positioning at x,y Coordinates.

Controlling the Write Direction

By default characters are written to the right. By using the `fm_fontdir` routine, you can write characters in any horizontal or vertical direction.

`fm_fontdir`*(gfd, direction)*

After calling this routine, any characters written to the display with the `fm_write` routine are written in the direction specified by the *direction* parameter. Valid values for *direction* are:

- `'u'`—upward,
- `'d'`—downward,
- `'l'`—to the left,
- `'r'`—to the right (this is the default).

This routine affects *only* the direction of the write and *not* the characters themselves—they are still displayed normally within each character cell. Also, the write direction stays in effect until a different font is activated; at that point, the direction returns to the default (`'r'`).

Example

The following example writes in all four directions. Remember that the x,y location specifies different corners of the character cell depending on the direction you specify. Because of this, the up, right and left directions can share the same origin. However, the down direction (or right direction) must have a different origin.

```
#include <starbase.c.h>                      /* get starbase defs             */

main()                                        /* program "FM_Directions.c"      */
{
    int  gfd;                                 /* file descriptor                */
    int  fid;                                 /* font id                        */

    if ((gfd = gopen("/dev/crt", OUTDEV, "hp98550", INIT)) == -1)
        exit(1);

    fm_load(gfd, "/usr/lib/raster/8x16/SNF/lp.8U.scf", &fid);

    fm_fontdir(gfd,'u');
    fm_write(gfd, 80, 100, "Upward", 6, TRUE, FALSE);
    fm_fontdir(gfd,'d');
    fm_write(gfd, 80, 120, "Down", 4, TRUE, FALSE);
    fm_fontdir(gfd,'r');
    fm_write(gfd, 80, 100, "Right", 5, TRUE, FALSE);
    fm_fontdir(gfd,'l');
    fm_write(gfd, 80, 100, "Left", 4, TRUE, FALSE);

    fm_remove(gfd, fid);                     /* to remove the font             */
    gclose(gfd);                             /* close the device               */

} /* end of main */
```

Setting Colors

To change the active font's foreground and background colors, call `fm_colors`.

`fm_colors`*(gfd, foreground, background)*

The *foreground* and *background* parameters specify the new colors to use; they are indexes into the Starbase color map.

Supported values are determined by the display device: 0 or 1 for monochromatic displays, 0 to 15 for 4-plane color, and 0 to 255 for 8-plane color.

The foreground and background colors return to the defaults (black and white) whenever a font is activated.

Example

The following code segment loads an 8×16-pixel bold font into the font cache, activates the font, changes its colors to black on white, writes the word "HELLO", and removes the font from the cache.

```
#include <starbase.c.h>                     /* get starbase defs          */
#define BLACK   0
#define WHITE   1

main()                             /* program "FM_Colors.c"      */
{
    int gfd;                                /* file descriptor            */
    int fid;                                /* font id                    */
    int i;
    if ((gfd = gopen("/dev/crt", OUTDEV, "hp98550", INIT)) == -1)
        exit(1);

    /* Load the font; the font is automatically activated */
    fm_load(gfd, "/usr/lib/raster/8x16/SNF/lp.b.8U.scf", &fid);

    for (i=0; i<8; i=i+2){
        fm_colors(gfd, i, i+1);
 fm_write(gfd, i*10, i*10, "HELLO", 5, TRUE, TRUE);
    }

    fm_remove(gfd, fid);                    /* Remove the font */
    gclose(gfd);                            /* Close the device           */
}
```

Optimizing Character Generation

Some systems have specialized hardware for writing to bit-mapped displays. This special hardware accelerates writing characters to the display. The `fm_opt` routine allows you to take advantage of specialized display hardware.

`fm_opt`*(gfd, optmode)*

If *optmode* is 1, optimization is turned on; if *optmode* is 0, optimization is turned off.

This routine fails if optimization hardware doesn't exist on the system or if too many fonts have been optimized already. This should not be considered a fatal error. Therefore, you should not abnormally terminate your program if this routine fails.

Determining String Length

To determine if a character string will extend beyond the edge of a window or display, use the `fm_str_len` routine to determine the pixel length of any character string along the current direction of the active font.

`fm_str_len`*(gfd, str, numchars)*

The *str* parameter points to the character string containing *numchars* characters.

This routine is especially useful if character clipping is disabled (see the "Character Clipping" section for details).

Example

The following example writes the message "HELLO" upwards and to the left. The origin of the string is determined by using the fm_str_len routine.

```
#include <starbase.c.h>                        /* get starbase defs         */

main()                                          /* program "FM_length.c"     */
{
    int gfd;                                    /* file descriptor           */
    int fid;                                    /* font id                   */
    int x,y;
    if ((gfd = gopen("/dev/crt", OUTDEV, "hp98550", INIT)) == -1)
        exit(1);

    fm_load(gfd, "/usr/lib/raster/8x16/SNF/1p.b.8U.scf", &fid);

    fm_fontdir(gfd, 'u');                       /* determine the length if writing */
    y = fm_str_len(gfd, "HELLO", 5);            /* upwards                         */
    fm_fontdir(gfd, 'l');                       /* determine the length if writing */
    x = fm_str_len(gfd, "HELLO", 5);            /* to the left                     */

    fm_fontdir(gfd, 'u');
    fm_write(gfd, x, y, "HELLO", 5, TRUE, FALSE);
    fm_fontdir(gfd, 'l');
    fm_write(gfd, x, y, "HELLO", 5, TRUE, FALSE);

    fm_remove(gfd, fid);                        /* Remove the font */
    gclose(gfd);                                /* Close the device          */
} /* end of main  */
```

Writing Sixteen-Bit Font Strings

The fm_sixteen_bit routine can be used to put the font manager in *sixteen_bit_mode*.

fm_sixteen_bit*(gfd, sixteen_bit_mode)*

If flag is 1, *sixteen_bit_mode* is turned on; if flag is 0, *sixteen_bit_mode* is turned off.

When the font manager is put into *sixteen_bit_mode*, character strings are interpreted 16 bits per character rather than the standard 8 bits per character.

This should not be confused with the use of HP-15 font files, which are parsed into mixed 8-bit and 16-bit portions. Rather, the 16-bit capability here would be used for writing pure 16-bit text. This is especially useful when using MIT 16-bit fonts. These fonts are not a mixture of 8-bit and 16-bit characters. They are all 16-bit characters.

Character Clipping

Character clipping controls the area in which characters are written. When character clipping is enabled, you cannot write characters outside the established clip limits. You can enable or disable clipping and redefine clip limits with font manager routines.

By default, when a graphics window is created, clipping is enabled and the clip limits are always set to the current window size. The default clip limits of a display are the phsical limits of the display.

Note Memory can become corrupted if clipping is not enabled. This is because characters could inadvertently be written outside the display memory established by the clip limits. Conceivably, you could write spurious data into your data structures and your program.

If you do not use clipping, be sure to check the length of every character string (with `fm_str_len`) to ensure that displaying the string will not cause it to extend outside the display boundaries.

Enabling/Disabling Clipping

To enable or disable clipping for a given display, use the `fm_clipflag` routine.

`fm_clipflag`*(gfd, flag)*

The *flag* parameter indicates whether to enable or disable clipping: if *flag* is `1`, clipping is enabled; if *flag* is `0`, clipping is disabled.

Setting Clip Limits

To set clip limits for a display, use the `fm_cliplim` routine.

> `fm_cliplim`*(gfd, x,y, width,height)*

The *x,y* parameters indicate the *x,y* location of the upper-left corner of the clipping rectangle (with respect to the upper-leftmost pixel of the display); the *width,height* parameters define the pixel width and height of the clipping rectangle. After calling this routine, characters can be written only within the defined rectangle.

Example

The following example writes "X"s across the window. It then sets the clipping area to a portion of the window and writes blanks to show the clipped area.

```
#include <starbase.c.h>                 /* get starbase defs          */

main()     /* program "FM_clipping.c"    */
{
    int gfd;                      /* file descriptor            */
    int fid;                            /* font id                    */

    if ((gfd = gopen("/dev/crt", OUTDEV, "hp98550", INIT)) == -1)
        exit(1);

    fm_load(gfd, "/usr/lib/raster/8x16/SNF/lp.8U.scf", &fid);

    fm_write(gfd, 10, 10, "XXXXXXXXXXXXXXXXXXXXXXXXX", 25, TRUE, FALSE);
    fm_write(gfd, 10, 20, "XXXXXXXXXXXXXXXXXXXXXXXXX", 25, TRUE, FALSE);
    fm_write(gfd, 10, 30, "XXXXXXXXXXXXXXXXXXXXXXXXX", 25, TRUE, FALSE);
    fm_write(gfd, 10, 40, "XXXXXXXXXXXXXXXXXXXXXXXXX", 25, TRUE, FALSE);
    fm_write(gfd, 10, 50, "XXXXXXXXXXXXXXXXXXXXXXXXX", 25, TRUE, FALSE);
    fm_write(gfd, 10, 60, "XXXXXXXXXXXXXXXXXXXXXXXXX", 25, TRUE, FALSE);
    fm_write(gfd, 10, 70, "XXXXXXXXXXXXXXXXXXXXXXXXX", 25, TRUE, FALSE);
    fm_write(gfd, 10, 80, "XXXXXXXXXXXXXXXXXXXXXXXXX", 25, TRUE, FALSE);
    fm_write(gfd, 10, 90, "XXXXXXXXXXXXXXXXXXXXXXXXX", 25, TRUE, FALSE);
    fm_cliplim(gfd, 25, 25, 50, 50);
    fm_clipflag(gfd, 1);

    fm_write(gfd, 10, 10, "                         ", 25, TRUE, FALSE);
    fm_write(gfd, 10, 20, "                         ", 25, TRUE, FALSE);
    fm_write(gfd, 10, 30, "                         ", 25, TRUE, FALSE);
    fm_write(gfd, 10, 40, "                         ", 25, TRUE, FALSE);
    fm_write(gfd, 10, 50, "                         ", 25, TRUE, FALSE);
    fm_write(gfd, 10, 60, "                         ", 25, TRUE, FALSE);
    fm_write(gfd, 10, 70, "                         ", 25, TRUE, FALSE);
    fm_write(gfd, 10, 80, "                         ", 25, TRUE, FALSE);

    fm_remove(gfd, fid);            /* Remove the font */
    gclose(gfd);                    /* Close the device           */
} /* end of main  */
```

Font Information Routines

The font manager library provides routines that obtain information about fonts. In particular you can inquire:

- Font size information.
- A font's path name.
- Information on a font's style.

Before discussing how to obtain font information, a discussion of font sizes and font styles is needed.

Font Size

Font size is actually comprised of three different attributes: width, height, and baseline height.

Font width and height are straightforward. Each character in a font is displayed in a font cell. The font cell is the same size for all characters. The font's width and height represent the pixel width and height of the font cell.

All the characters of a given font "sit" on an invisible line called the **baseline.** The "bottom" of each character is flush with this line. However, parts of characters can extend below the baseline—for example, the descender that extends below the circle on the letter p. The following figure illustrates each of these size attributes.

The baseline attribute allows you to align different-sized fonts on the same line. For example, suppose you are writing a story that starts with "In the beginning," and you want the first letter, I, to be in a large font and the rest of the characters to be in a normal-sized font. To make the text look more natural, you should align the baseline of the big I with the baseline of the normal-sized font.

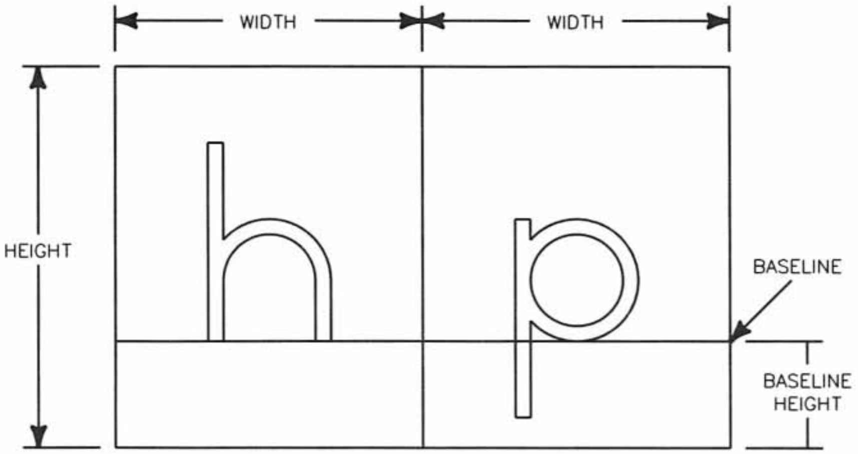

Figure 3–2. Font Size Attributes.

Getting Font Size Information

Two font manager routines return font size information: `fm_fileinfo` and `fm_rasterinfo`. The `fm_rasterinfo` routine gets size information for fonts in the font table.

`fm_rasterinfo`*(gfd, fontid, width, height, baseline)*

This routine returns the font cell width and height (in pixels) and the baseline height (also in pixels) for the font specified by *fontid*.

The `fm_fileinfo` routine gets size information for a font file.

`fm_fileinfo`*(path, width, height, baseline)*

This routine returns font size information for the font file whose path name is pointed to by *path*.

Note This routine does not accept a *gfd* parameter and has no way of determining which file system fonts are being loaded from. Thus, it is assumed that the font path specifies a font file on the local file system.

Example

The following example writes the phrase “In the beginning” to a window. The “I” is in a larger font than the rest of the phrase. To line up the rest of the phrase, it is necessary to know the size of the first font used.

```
#include <starbase.c.h>            /* get starbase defs            */

main()                              /* program "Font_size.c"        */
{
    int  gfd;                       /* file descriptor              */
    int  fid, fid2;                 /* font id                      */
    int big_width, big_height;      /* big font cell size           */
    int big_baseline;               /* big font baseline            */
    int small_width, small_height;  /* small font cell size         */
    int small_baseline;             /* small font baseline          */
    int offset;                     /* change in font size          */

    if ((gfd = gopen("/dev/crt", OUTDEV, "hp98550", INIT)) == -1)
       exit(1);

    fm_load(gfd, "/usr/lib/raster/18x30/SNF/pica.8U.scf:, &fid);
    fm_write(gfd, 10, 80, "I", 1, TRUE, TRUE);
    fm_rasterinfo(gfd, fid, &big_width, &big_height, &big_baseline);

    fm_load(gfd, "/usr/lib/raster/10x20/SNF/lp.8U.scf", &fid2);
    fm_rasterinfo(gfd, fid2, &small_width, &small_height,
        &small_baseline);
    offset = big_height - big_baseline - (small_height - small_baseline);
    fm_write(gfd, 10 + big_width, 80 + offset, "n the beginning", 15, TRUE,
        TRUE);

    fm_remove(gfd, fid);                /*Remove the fonts              */
    fm_remove(gfd, fid2);
    gclose(gfd);                        /* close the device             */

} /* end of main */
```

Font Style

Each font has certain attributes that define its style. These attributes are defined by the `escapecodes` structure in the *fonticon.h* header file. The following table briefly defines each of the fields in this structure.

Table 3–1.

Item	Description	Range
`symbol_int`	Gives the numerical part of the font's identification string; e.g., 8 for 8-bit Roman-8 (8U); 0 for 7-bit math font (0M). The value for this field indicates whether the font is 8-bit (=8) or 7-bit (=0).	0, 7, or 8
`typeface`	Specifies the kind of typeface, e.g. pica=1, prestige=8, etc.	0 to 10
`proportional`	Tells whether the font is uniform width (=0) or proportional (=1).	0 or 1
`hpitch`	Approximates horizontal characters per inch.	Depends on font width.
`vheight`	Approximates vertical characters per inch.	Depends on font height.
`boldness`	Indicates the boldness of the font. The lightest is -7, the boldest is 7.	-7 to 7
`quality`	Describes the quality of the font: data processing (=0), near letter quality (=1), or correspondence quality (=2).	0 to 2

Getting Font Style Information

To get font style information, call the `fm_styleinfo` routine.

`fm_styleinfo`*(gfd, fontid, symbol_char, escapecodes)*

The *symbol_char* parameter returns a character describing the font (that is, `'U'` for Roman-8 fonts, `'K'` for Katakana fonts).

The *escapecodes* parameter returns an `escapecodes` structure as defined in *fonticon.h*. This structure contains style information for the font indicated by *fontid*.

The Windows/9000 fonts all contain the necessary information to surpport the `fm_styleinfo` routine, but not all SNF fonts contain this information. If for any reason, any part of this information is missing from the font, a value of `-1` will be returned in the corresponding field.

Getting a Font's Name

The `fm_getfontid` routine translates a font id into its corresponding font name.

> `fm_getname`*(gfd, fontid, filename)*

The *filename* parameter returns a character string containing the pathname used to load the font represented by *fontid*.

Example

The following function gets font size, style, and name information for the font specified by the *fid* parameter; it returns this information to the calling program.

```
#include <fonticon.h>                    /* font manager definitions       */
#include <starbase.c.h>                  /* get starbase defs              */

main()              /* program "Style.c"              */
{
    int  gfd;                            /* file descriptor                 */
    int  fid;                            /* font id                         */
    struct escapecodes esc;              /* font-specific style information */
    char symbol_char;                    /* character desribing the font    */
    char pathname[80];                   /* pathname of the font       */

    if ((gfd = gopen("/dev/crt", OUTDEV, "hp98550", INIT)) == -1)
        exit(1);

    fm_load(gfd, "/usr/lib/raster/18x30/SNF/pica.8U.scf", &fid);
    fm_getname(gfd, fid, pathname);
    printf("For the font in %s\n", pathname);
    fm_styleinfo(gfd, fid, &symbol_char, &esc);
    printf("The symbol_char is %c\n", symbol_char);
    printf("The escape codes are:\n");
    printf("          symbol_int: %d\n", esc.symbol_int);
    printf("            typeface: %d\n", esc.typeface);
    printf("        proportional: %d\n", esc.proportional);
    printf("              hpitch: %d\n", esc.hpitch);
    printf("             vheight: %d\n", esc.vheight);
    printf("               style: %d\n", esc.style);
    printf("            boldness: %d\n", esc.boldness);
    printf("             quality: %d\n", esc.quality);

    fm_remove(gfd, fid);                /* Remove the font                  */
    gclose(gfd);                        /* Close the device                 */
} /* end of main */
```

4

Compiling FA/FM Programs

Linking Window Libraries

When compiling a program that calls window, fast alpha, or font manager library routines, link the libraries in the order shown below. You only need to link the libraries that the program uses.

1. `libfa.a`—if the program calls any fast alpha routines, link this library first.
2. `libfontm.a`—if the program calls any fast alpha or font management routines, link this library.
3. `libdd`*driver*`.a`—always link the device *driver* (or drivers) of the display on which the programs run. Not all starbase drivers support FA/FM. See the *Starbase Device Drivers Library* manual to determine which drivers to use.
4. `libddbyte.a` or `libddbit.a`—if the program performs graphics to windows with retained rasters, and you want the raster to be maintained in memory, load this driver, which writes to the retained memory. If the program does not use windows, you do not need to link this library.
5. `libXwindow.a`, `libXhp11.a`, `libXr11.a`, `libX11.a`—these libraries should be linked if the program calls X window routines or performs graphics (fast alpha, font manager) output to an X window. If the program does not use X windows, you do not need to link these libraries.
6. `libwindow.a`—this library should be linked if the program calls HP Windows/9000 window routines or performs graphics (fast alpha, font manager) output to an X window. If the program does not use HP Windows/9000 or X11 windows, you do not need to link this library.
7. `libsb1.a`—link this library if the program calls any Starbase graphics, fast alpha, or font manager routines.

8. `libsb2.a`—link this library immediately after `libsb1.a` if `libsb1.a` was loaded.

Examples

The following examples should help clarify how the libraries are linked with the main program.

A C program named `xpg.c` that creates and manipulates graphics windows calls only font manager routines, and performs starbase graphics to a retained X11 window on an HP 98730 display would be compiled as:

```
cc xpg.c -lfontm -ldd98730 -lddbyte -lXwindow -lsb1 -lsb2 -lXhp11 -lX11
```

A C program named `faprog.c` that creates and manipulates graphics windows, calls fast alpha and font management routines, and performs graphics to a retained HP Windows/9000 graphics window on an HP 98700 display would be compiled as:

```
cc faprog.c -lfa -lfontm -ldd98700 -lddbyte -lwindow -lsb1 -lsb2
```

A C program named `gpr.c` that calls fast alpha and font management routines and performs graphics on an HP 98720 display would be compiled as:

```
cc gpr.c -lfa -lfontm -ldd98720 -lsb1 -lsb2
```

Font and Icon Files

`/usr/lib/raster/*`	Contains all font directories.
`/usr/lib/raster/icons`	Icon definition files are stored here.
`/usr/lib/raster/dflt/b/h/$LANG`	If this file is present, it is the default base font (`/usr/lib/raster/dflt`), (`/b`) for high-resolution displays (`/h`) for the language defined by the `$LANG` environment variable.

`/usr/lib/raster/dflt/b/l/$LANG`	If this file is present, it is the default base font (`/usr/lib/raster/dflt`), (`/b`) for low-resolution displays (`/l`) for the language defined by the `$LANG` environment variable.
`/usr/lib/raster/dflt/a/h/$LANG`	If this file is present, it is the default alternate font (`/usr/lib/raster/dflt`), (`/a`) for high-resolution displays (`/h`) for the language defined by the `$LANG` environment variable.
`/usr/lib/raster/dflt/a/l/$LANG`	If this file is present, it is the default alternate font (`/usr/lib/raster/dflt`), (`/a`) for low-resolution displays (`/l`) for the language defined by the `$LANG` environment variable.

Note The actual SNF versions of these font files are in SNF subdirectories.

Header Files

There are two files that contain information about the fast alpha and font manager structures.

`/usr/include/fa.h`	Fast alpha constant and structure definitions.
`/usr/include/fonticon.h`	Font constant and structure definitions.

COMMAND SUMMARY

fa	summary of fast alpha library routines
faclear	clear the window area specified by the given rectangle
facolors	set the fast alpha font foreground and background colors
facursor	control the displayed cursor
fafontactivate	activate a fast alpha font
fafontload	load a font into the user´s font cache and prepare it for activation
fafontremove	remove a font from the user´s font cache
fagetinfo	get information about the fast alpha environment
fainit	prepare a fast alpha window device for output, and set up all defaults
farectwrite	fill an area of the window with the specified character
faroll	roll a portion of the window
fasetinfo	set information about the fast alpha environment
faterminate	terminate the current fast alpha environment
fawrite	write a line of characters with their enhancements
fm_activate	make a font active
fm_clipflag	set clipping flag
fm_cliplim	set clip limits
fm_colors	set active font´s foreground and background colors
fm_fileinfo	return the size of cells in a font file
fm_fontdir	set character direction
fm_getname	translate font id to filename
fm_load	load a font into memory
fm_opt	optimize character generation if possible
fm_rasterinfo	return the size of cells in a font
fm_remove	remove a font
fm_sixteen_bit	set sixteen bit flag
fm_str_len	determine the pixel length of a character string
fm_styleinto	return style information about a font
fm_write	write characters to the screen
fontm	summary of font manager library routines

NOTES

NAME

fa – summary of fast alpha library routines

DISCUSSION

The fast alpha library, **/usr/lib/libfa.a**, provides high-performance alpha (textual) capabilities with graphics windows and bitmapped displays. Fast alpha assumes uniform width fonts and provides a row and column interface. Fast alpha also implements the concepts of a cursor and display enhancements.

All fast alpha library calls require a *gopen(3G)* file descriptor that may be for a bitmapped display device or for a graphics window.

Programs that call fast alpha routines must link in the fast alpha library (-lfa). In addition, programs that call fast alpha library routines must also link in the font manager (-lfontm) and Starbase (-lsb1 -lsb2) libraries.

The header file **/usr/include/fa.h** contains structure and constant definitions used by fast alpha library routines. Programs should attempt to use these definitions when calling fast alpha library routines.

Fast alpha library routines are summarized below. For more information on each routine, consult its reference page.

faclear(3W)	Clear a window area specified by a *fa_rectangle* structure, defined in **fa.h**.
facolors(3W)	Set the font foreground and background colors for the currently activated fast alpha font.
facursor(3W)	Position and enable/disable the fast alpha cursor.
fafontactivate(3W)	Activate a fast alpha font.
fafontload(3W)	Load a font file into the font cache and activate it.
fafontremove(3W)	Remove a font from the font cache.
fagetinfo(3W)	Get information about the current fast alpha environment. Return this information in an *fainfo* structure, defined in **fa.h**.
fainit(3W)	Prepare a fast alpha window for output; set the window's fast alpha environment to default values.
farectwrite(3W)	Fill an area of a window, defined by an *fa_rectangle* structure from **fa.h**, with a character.
faroll(3W)	Roll (scroll) a portion of a window. The area to roll is defined by an *fa_rectangle* structure from **fa.h**.
fasetinfo(3W)	Set information about the fast alpha environment. Information is set from an *fainfo* structure, defined in **fa.h**.
faterminate(3W)	Terminate the current fast alpha environment, which was initialized via *fainit*(3W).
fawrite(3W)	Write characters in the active font.

EXAMPLES

The following example compiles a program, named *fawinprog.c*, that calls fast alpha routines to display text in graphics windows. The program will run on a Series 300 high-resolution display using retained graphics windows (thus the -ldd300h and -lddbyte options).

```
cc fawinprog.c -lfa -lfontm -ldd300h -lddbyte -lwindow -lsb1 -lsb2
```

SEE ALSO

windows(1),fontm(3W),gopen(3G),window(3W).

NAME

faclear – clear the window area specified by the given rectangle

SYNOPSIS

```
#include <fa.h>
int faclear(gfd,reserved,rp);
int gfd;
int reserved;
struct fa_rectangle *rp;
```

DESCRIPTION

gfd	is an integer file descriptor for an *gopen*ed device interface.
reserved	this field is reserved for future expansion
rp	is the pointer to the data structure which describes the rectangle to be cleared.

DISCUSSION

This routine clears the area bounded by the rectangle pointed to by *rp.*

By default, this routine clears the screen immediately. It is possible to get better performance by buffering clear operations and updating the screen after several operations are buffered. See *fasetinfo*(3W) and the **fa.h** header file for information on how to buffer-up writes, as opposed to having them occur when this routine is called.

HARDWARE DEPENDENCIES

Series 500:

Fast alpha routines do not support HP-15 (2-byte) fonts on Series 500.

SEE ALSO

fagetinfo(3W).

DIAGNOSTICS

A value of -1 is returned if *gfd* is invalid or a call to *fainit* was never executed on this *gfd.* See *errno*(2) for further information.

NAME

facolors – set the fast alpha font foreground and background colors

SYNOPSIS

#include <fa.h>
int facolors(gfd,foreground,background);
int gfd;
int foreground,background;

DESCRIPTION

gfd	is an integer file descriptor for an *gopen*ed device interface.
foreground	is the new foreground color.
background	is the new background color.

DISCUSSION

Sets the foreground and background colors of the activated font for all further fast alpha operations. These colors are indices into the system color map. Valid values are: 0 and 1 for monochromatic displays, 0 to 15 for 4-plane color, and 0 to 255 for 16-plane color. The color table index 0 is assumed to be black and the index 1 is assumed to be white. Note, that if *colormode* is not set to FACOLOR, the system will ignore any color changes; see *fagetinfo* and *fasetinfo.*

Color values outside the acceptable range (i.e., 0 or 1 for monochrome, 0 to 15 for 4-plane color, and 0 to 255 for 8-plane color) will default to 1 (white).

HARDWARE DEPENDENCIES

Series 500:

Fast alpha routines do not support HP-15 (2-byte) fonts on Series 500.

SEE ALSO

fasetinfo(3W), fagetinfo(3W).

DIAGNOSTICS

A value of -1 is returned if *gfd* is invalid or a call to *fainit* was never executed on this *gfd.* See *errno*(2) for further information.

NAME

facursor – control the displayed cursor

SYNOPSIS

#include <fa.h>
int facursor(gfd,column,line,cflag);
int gfd;
int column,line;
int cflag;

DESCRIPTION

gfd	is an integer file descriptor for an *gopen*ed device interface.
column	indicates the alpha column at which to do the operation.
line	indicates the alpha line at which to do the operation.
cflag	if *cflag* is TRUE, the cursor is made visible. If FALSE, the cursor is made invisible (if possible). If *column* and *line* are valid coordinates, the cursor (visible or invisible) is positioned accordingly. If either is invalid (i.e. -1), the cursor position is not affected. FACURSORNOMOVE is defined as -1 in **/usr/include/fa.h** to provide a mnemonic for specifying invalid values. This is useful for updating the cursor position without actually moving it.

DISCUSSION

This routine allows the user to position and turn on and off the cursor. It also allows the user to "store" or change a cursor position without moving its physical position until a later operation.

The cursor size is the character cell size of a 1-byte character even if a HP-15 (2-byte) font is used. The 2-byte characters are twice as wide as the 1-byte characters. Thus they take up two columns.

By default, fast alpha "writes" update the screen immediately. It is possible to get better performance by buffering writes and updating the screen **after** the writes are buffered. See *fasetinfo*(3W) and the **fa.h** header file for information on how to buffer-up writes, as opposed to having them occur when this routine is called.

HARDWARE DEPENDENCIES

Series 500:

Fast alpha routines do not support HP-15 (2-byte) fonts on Series 500.

SEE ALSO

fawrite(3W).

DIAGNOSTICS

A value of -1 is returned if *gfd* is invalid or a call to *fainit* was never executed on this *gfd*. See *errno*(2) for further information.

NAME

fafontactivate – activate a fast alpha font

SYNOPSIS

int fafontactivate(gfd,fontid);
int gfd, fontid;

DESCRIPTION

gfd	is an integer file descriptor for an *gopen*ed device interface.
fontid	is the id of the font to activate.

DISCUSSION

This routine sets the specified font as the currently active font for the current window device. The specified font must be one that was previously made available by *fafontload* or *fm_load*.

HARDWARE DEPENDENCIES

Series 500:

Fast alpha routines do not support HP-15 (2-byte) fonts on Series 500.

SEE ALSO

fafontload(3W), fm_load(3W).

DIAGNOSTICS

A value of -1 is returned if *gfd* or *fontid* is invalid or a call to *fainit* was never executed on this *gfd*. See *errno*(2) for further information.

NAME

fafontload – load a font into the user's font cache and prepare it for activation

SYNOPSIS

```
int fafontload(gfd,path);
int  gfd;
char *path;
```

DESCRIPTION

gfd	is an integer file descriptor for an *gopen*ed device interface.
path	is the path name of the font to be loaded.

DISCUSSION

This routine loads a font into the user's cache of available fonts. Any of these fonts are ready to be activated as needed. A system-wide unique font id is returned unless *gfd or path* are invalid, in which case -1 is returned. Like *fm_fontload*(3W), *fafontload* automatically makes the font active when it is loaded. Note that font manager and fast alpha font ids are the same and can be used with both fast alpha and font manager routines.

This routine will try to optimize the font if the font file header block indicates to do so.

HARDWARE DEPENDENCIES

Series 500:

Fast alpha routines do not support HP-15 (2-byte) fonts on Series 500.

SEE ALSO

fafontremove(3W), fafontactivate(3W), fm_load(3W).

DIAGNOSTICS

A value of -1 is returned if gfd is invalid or a call to *fainit* was never executed on this *gfd.* See *errno*(2) for further information.

WARNING

When using either rectangular fonts (pixelformat = 1) or HP-15 fonts (pixelformat = 2) with a low resolution display and retained rasters, the portion that gets obscured will lose every other pixel of information. Also, 2-byte characters that are written to the obscured portion of the raster will appear twice the size they should be when the area is unobscured.

NAME

fafontremove – remove a font from the user's font cache

SYNOPSIS

int fafontremove(gfd,fontid);
int gfd,fontid;

DESCRIPTION

gfd	is an integer file descriptor for an *gopen*ed device interface.
fontid	a unique system id assigned to a font when the font is loaded with *fafontload.*

DISCUSSION

This routine deletes the font from the user's font cache. If this font was the currently active font, an error will occur if further fast alpha operations are attempted without first activating another font.

HARDWARE DEPENDENCIES

Series 500:

Fast alpha routines do not support HP-15 (2-byte) fonts on Series 500.

SEE ALSO

fafontload(3W), faterminate(3W).

DIAGNOSTICS

A value -1 is returned if *gfd* or *fontid* is invalid or a call to *fainit* was never executed on this *gfd.* See *errno*(2) for further information.

NAME

fagetinfo – get information about the fast alpha environment

SYNOPSIS

#include <fa.h>
int fagetinfo(gfd,fainfoptr)
int gfd;
struct fainfo *fainfoptr;

DESCRIPTION

gfd is an integer file descriptor for a *gopen*ed device interface.

fainfoptr is a pointer to the structure defined as follows:

```
struct fainfo {
        struct fa_rectangle size;
        int capabilities;
        int enhancements;
        int defaultenhancements;
        int cursor;
        int fontcellheight;
        int fontcellwidth;
        int foregroundplanes;
        int backgroundplanes;
        int clearbeforewrite;
        int colormode;
        int makecurrent;
};
```

Descriptions of each field in this structure are:

size a structure with one corner set to [0, 0] and the other set to [number of columns, number of lines]. (See Fast Alpha Rectangles in the *Fast Alpha/Font Manager Programmer's Manual.*)

capabilities
an integer assigned a value from a list in the header file. The contents of capabilities may be used by a program to detect what additional capabilities are available on a particular device. Currently defined capabilities are:

FAWINDOW - Device is a window or bitmapped graphics device. More information may be obtained by calls to other window system routines.

enhancements
is assigned to the bitwise ORing of the various enhancement bits (see *farectwrite*) which are supported by the particular device.

defaultenhancements
is initially set to a value which optimizes the performance of the window system. It is used by *fawrite* and *faroll.* It is a read/write field in that it can be changed by calling *fasetinfo.*

cursor cursor is TRUE if the cursor may be physically removed from the window device and is FALSE otherwise.

fontcellheight, fontcellwidth
indicates the size (in pixels) of the fontcell of 1-byte characters. The 2-byte characters are twice as wide. The cursor size and the column addressing are based on the size of 1-byte characters.

clearbeforewrite
is TRUE if the fast alpha library routine is to clear the background of the area to be written to before the characters are written. *clearbeforewrite* set to FALSE indicates that the user is responsible for clearing and desires no background clearing by the library routine. *clearbeforewrite* defaults to TRUE. *clearbeforewrite* currently pertains only to FACOLOR mode; see colormode, below.

foregroundplanes, backgroundplanes
specifies the number of memory planes available for controlling the foreground and background colors respectively. A value of 1 in *foregroundplanes* and 1 in *backgroundplanes* indicates a monochrome system. Values > 1 indicate a color system.

colormode
when set to FAWONB (white on black - which is the default) indicates that the user is not using the color options. This allows the fast alpha routines to run somewhat faster than when color is incorporated. When *colormode* is set to FACOLOR, the additional system operations needed to incorporate color are performed. A third option is to set *colormode* to FABONW (black on white) which inverses the previous monochromatic option, FAWONB; see *facolors.*

makecurrent
setting this field to bitwise ORing of values specified in the header file (**/usr/include/fa.h**) controls the appearance of fast alpha operations on the screen. For performance reasons, the user may choose to suppress the updating of the screen until several operations are queued up. When updating is desired, the user signals the fast alpha environment to update the screen by setting the *makecurrent* value to MCALWAYS (make current always). Queued up operations will show up on the screen at this time. The default value is MCALWAYS, that is, update the screen upon every fast alpha operation.

DISCUSSION

This routine is used to find out information about the window device, or to get currently set values in the fast alpha environment. Display information might include the size of the window device and the video attributes it supports. Fast alpha environment values might include the size of the current window device or the size of the currently active font.

fagetinfo is the counterpart of *fasetinfo.* It is used for inquiring the values of parameters in a fast alpha environment.

HARDWARE DEPENDENCIES

Series 500:

Fast alpha routines do not support HP-15 (2-byte) fonts on Series 500.

SEE ALSO

fasetinfo(3W), fainit(3W).

DIAGNOSTICS

A value of -1 is returned if *gfd* is invalid or a call to *fainit* was never executed on this *gfd.* See *errno*(2) for further information.

NAME

fainit – prepare a fast alpha window device for output, and set up all defaults

SYNOPSIS

#include <fa.h>
int fainit(gfd,driver);
int gfd;
int driver;

DESCRIPTION

gfd	is an integer file descriptor for an *gopen*ed device interface.
driver	indicates the driver to be used for all subsequent fast alpha calls, i.e., indicates a particular display device. See **/usr/include/fa.h** for all valid driver names. The value FAWINDOW should be used for HP Windows/9000 Fast Alpha Library.

DISCUSSION

This routine is used to set up the default values the fast alpha environment will use for future fast alpha library calls. It assumes that a *gopen*(3S) has actually been performed, and that the returned file descriptor is supplied as *gfd*.

These variables remain in effect until font or color changing library routines are invoked or until an *faterminate* is encountered. Calls to *fasetinfo* with proper parameters can alter the state set up by *fainit*.

HARDWARE DEPENDENCIES

Series 500:

Fast alpha routines do not support HP-15 (2-byte) fonts on Series 500.

SEE ALSO

faterminate(3W), fasetinfo(3W).

DIAGNOSTICS

A value of -1 is returned if *gfd* or *driver* is invalid or if *fainit* otherwise fails. See *errno*(2) for further information.

NAME

farectwrite – fill an area of the window with the specified character

SYNOPSIS

#include <fa.h>
int farectwrite(gfd,character,enhancement,rp);
int gfd;
int character;
int enhancement;
struct fa_rectangle *rp;

DESCRIPTION

gfd	is an integer file descriptor for an *gopen*ed device interface
character	is the character to be used to fill the given rectangle area. Only 1-byte character codes may be used; 2-byte characters cannot be used for the rectangle fill.
enhancement	is the enhancement to be used in the rectangle fill operation.
rp	is the pointer to the data structure which describes the rectangle to be filled.

DISCUSSION

This routine writes what is defined by character and enhancement to all positions in the area bounded by the rectangle pointed to by *rp*.

Legal enhancements are:

FAOFF
FAINVERSE
FAUNDERLINE

The set of enhancements supported by a particular device should first be determined by calling *fagetinfo*. Legal enhancement values are formed from the bitwise ORing of the desired enhancements which are supported by the particular device. If no enhancement is given, i.e., enhancement is 0 (zero), *defaultenhancements*, from *fasetinfo*, is used. The user may indicate that no enhancements are desired by using FAOFF for the enhancement.

By default, fast alpha "writes" update the screen immediately. It is possible to get better performance by buffering writes and updating the screen **after** the writes are buffered. See *fasetinfo*(3W) and the **fa.h** header file for information on how to buffer-up writes, as opposed to having them occur when this routine is called.

HARDWARE DEPENDENCIES

Series 500:

Fast alpha routines do not support HP-15 (2-byte) fonts on Series 500.

SEE ALSO

fagetinfo(3W), fasetinfo(3W).

DIAGNOSTICS

A value of -1 is returned if *gfd* is invalid or a call to *fainit* was never executed on this *gfd*. See *errno*(2) for further information.

NAME

faroll – roll a portion of the window

SYNOPSIS

```
#include <fa.h>
int faroll(gfd,how,howfar,rp);
int gfd;
int how;
int howfar;
struct fa_rectangle *rp;
```

DESCRIPTION

gfd is an integer file descriptor for an *gopen*ed device interface.

how determines the direction the window is to be rolled. Valid directions are:

FAROLLUP	'u'
FAROLLDOWN	'd'
FAROLLEFT	'l'
FAROLLRIGHT	'r'

howfar
the window is rolled *howfar* character units in the direction given by *how.*

rp is a pointer to the data structure which describes the rectangle to be filled.

DISCUSSION

This routine rolls the area bounded by the rectangle described by *rp.*

Any enhancements present are also rolled.

The area "uncovered" by the roll is filled as if *farectwrite* were called given the space character and *defaultenhancements* as parameters, see *farectwrite.* This routine does not affect the cursor.

When 2-byte characters are displayed and the rolling is done, some 2-byte characters may be split into two parts, or half of the character may be erased. This is because the rolling is done based on columns, and each 2-byte character takes up two columns.

By default, the fast alpha rolling operation updates the screen immediately. It is possible to get better performance by buffering roll operations and updating the screen **after** the roll operations are buffered. See *fasetinfo*(3W) and the **fa.h** header file for information on how to buffer-up rolls, as opposed to having them occur when this routine is called.

HARDWARE DEPENDENCIES

Series 500:

Fast alpha routines do not support HP-15 (2-byte) fonts on Series 500.

SEE ALSO

farectwrite(3W).

DIAGNOSTICS

A value of -1 is returned if *gfd* is invalid or a call to *fainit* was never executed on this *gfd.* See *errno*(2) for further information.

NAME

fasetinfo – set information about the fast alpha environment

SYNOPSIS

#include <fa.h>
int fasetinfo(gfd,fainfoptr);
int gfd;
struct fainfo *fainfoptr;

DESCRIPTION

gfd is an integer file descriptor for a *gopen*ed device interface.

fainfoptr is a pointer to the structure defined as follows:

```
struct fainfo {
        struct fa_rectangle size;
        int capabilities;
        int enhancements;
        int defaultenhancements;
        int cursor;
        int fontcellheight;
        int fontcellwidth;
        int foregroundplanes;
        int backgroundplanes;
        int clearbeforewrite;
        int colormode;
        int makecurrent;
};
```

Descriptions of each field in this structure are:

size a structure with one corner set to [0, 0] and the other set to [number of columns, number of lines]. (See Fast Alpha Rectangles in the *Fast Alpha/Font Manager Programmer's Manual.*)

This parameter cannot be set by *fasetinfo*. It is used to obtain information via *fagetinfo*.

capabilities
an integer assigned a value from a list in the header file. The contents of capabilities may be used by a program to detect what additional capabilities are available on a particular device. Currently defined capabilities are:

FAWINDOW - Device is a window or bitmapped graphics device. More information may be obtained by calls to other window system routines.

This parameter cannot be set by *fasetinfo*. It is used to obtain information via *fagetinfo*.

enhancements
is assigned to the bitwise ORing of the various enhancement bits (see *farectwrite*) which are supported by the particular device.

defaultenhancements
is initially set to a value which optimizes the performance of the window system. It is used by *fawrite* and *faroll*. It is a read/write field in that it can be changed by calling *fasetinfo*.

cursor cursor is TRUE if the cursor may be physically removed from the window device and is FALSE otherwise.

This parameter cannot be set by *fasetinfo*. It is used to obtain information via *fagetinfo*.

fontcellheight, fontcellwidth
indicates the size (in pixels) of the 1-byte character fontcell. The 2-byte characters are twice as wide.

This parameter cannot be set by *fasetinfo*. It is used to obtain information via *fagetinfo*.

clearbeforewrite
is TRUE if the fast alpha library routine clears the background of the area to be written to before the characters are written. *clearbeforewrite* set to FALSE indicates that the user is responsible for clearing and desires no background clearing by the library routine. *clearbeforewrite* defaults to TRUE. *clearbeforewrite* currently pertains only to FACOLOR mode; see colormode, below.

foregroundplanes, backgroundplanes
specifies the number of memory planes available for controlling the foreground and background colors respectively. A value of 1 in *foregroundplanes* and 1 in *backgroundplanes* indicates a monochrome system. Values > 1 indicate a color system.

This parameter cannot be set by *fasetinfo*. It is used to obtain information via *fagetinfo*.

colormode
when set to FAWONB (white on black - which is the default) indicates that the user is not using the color options. This allows the fast alpha routines to run somewhat faster than when color is incorporated. When *colormode* is set to FACOLOR, the additional system operations needed to incorporate color are performed. A third option is to set *colormode* to FABONW (black on white) that inverses the previous monochromatic option, FAWONB; see *facolors*.

makecurrent
setting this field to bitwise ORing of values specified in the header file (**/usr/include/fa.h**) controls the appearance of fast alpha operations on the screen. For performance reasons, the user may choose to suppress the updating of the screen until several operations are queued up. When updating is desired, the user signals the fast alpha environment to update the screen by setting the *makecurrent* value to MCALWAYS (make current always). Queued up operations will show up on the screen at this time. The default value is MCALWAYS, that is, update the screen upon every fast alpha operation.

DISCUSSION

This routine is used to set information regarding the window device, or to set new values in the fast alpha environment. Parameters which are user–settable include: *defaultenhancements*, *clearbeforewrite*, *makecurrent*, and *colormode*.

fasetinfo is the counterpart of *fagetinfo*. It is used for changing certain parameters of a fast alpha environment.

HARDWARE DEPENDENCIES

Series 500:
Fast alpha routines do not support HP-15 (2-byte) fonts on Series 500.

SEE ALSO

fagetinfo(3W).

DIAGNOSTICS

A value of -1 is returned if *gfd* is invalid or a call to *fainit* was never executed on this *gfd*. See *errno*(2) for further information.

NAME

faterminate – terminate the current fast alpha environment

SYNOPSIS

int faterminate(gfd)
int gfd;

DESCRIPTION

gfd	is an integer file descriptor for an *gopen*ed device interface.

DISCUSSION

This routine is the opposite of *fainit.* It frees up all system resources acquired during *fainit* and discards all fast alpha state information associated with the current file descriptor.

faterminate has no effect on the currently active font. It remains active unless a *fafontremove* on the active font was encountered prior to *faterminate.*

HARDWARE DEPENDENCIES

Series 500:

Fast alpha routines do not support HP-15 (2-byte) fonts on Series 500.

SEE ALSO

fainit(3w), fafontremove(3w), fafontactivate(3w).

DIAGNOSTICS

A value of -1 is returned if *gfdf* is invalid or a call to *fainit* was never executed on this *gfd.* See *errno*(2) for further information.

NAME

fawrite – write a line of characters with their enhancements

SYNOPSIS

```
#include <fa.h>
int fawrite(gfd,column,line,charbuf,ebuf,strlen);
int gfd;
int column, line;
char *charbuf;
ENH *ebuf;
int strlen;
```

DESCRIPTION

gfd	is an integer file descriptor for an *gopen*ed device interface.
column	indicates the alpha column at which to start the operation.
line	indicates the alpha line at which to do the operation.
charbuf	points to the characters to write (a null character does **not** terminate this string!)
ebuf	points to the corresponding enhancements for each character (i.e. the third character in *charbuf* receives the enhancement of the third element of *ebuf*). *ebuf* may be NULL signifying that no enhancements are indicated. In this case, the *defaultenhancements* (from *fasetinfo)* is used. The valid values for *ebuf* are taken from the set described in *farectwrite*. Enhancement values are the bitwise ORing of the values supported by the particular device. Previously enhanced character positions lose their enhancement when overwritten by a non-enhanced character (i.e. *ebuf* is NULL).
strlen	is the number of bytes to write. If the string contains 2-byte characters, each 2-byte character is counted as two bytes.

DISCUSSION

This library call writes *strlen* characters on the window device starting at the locations described by *column* and *line*.

fawrite does not affect the cursor position.

Attempts to use *fawrite* for wrapping or scrolling may produce undesired results.

The following processing is done only when the current active font is a HP-15 (2-byte) character font:

* If the string to be output contains an undefined 2-byte character, the 2-byte galley character is output instead of the undefined character.
* If the string contains an illegal 2-byte character, it outputs the two 1-byte characters that correspond to the code values of the illegal 2-byte character.
* If the last code in the string is the first code of a 2-byte character, it outputs the 1-byte character that corresponds to the code value.

By default, fast alpha "writes" update the screen immediately. It is possible to get better performance by buffering writes and updating the screen **after** the writes are buffered. See *fasetinfo*(3W) and the **fa.h** header file for information on how to buffer-up writes, as opposed to having them occur when this routine is called.

HARDWARE DEPENDENCIES

Series 500:

Fast alpha routines do not support HP-15 (2-byte) fonts on Series 500.

SEE ALSO

fasetinfo(3W), farectwrite(3W), facursor(3W).

DIAGNOSTICS

A value of -1 is returned if *gfd* is invalid or a call to *fainit* was never executed on this *gfd.* See *errno*(2) for further information.

BUGS

If a rectangular font (pixelformat = 1) or a HP-15 (2-byte) font (pixelformat = 2) is used on a low-resolution display with retained rasters, every other rectangular pixel will be lost when an area is obscured. Also, 2-byte characters that are written to obscured area will be twice as big as they should be when the area is displayed.

NAME

fm_activate – make a font active

SYNOPSIS

```
int  fm_activate(gfd,fontid);
int gfd, fontid;
```

DESCRIPTION

gfd	is an integer file descriptor for an *gopen*ed device interface.
fontid	is the id of the font to activate.

DISCUSSION

Sets the specified raster font as the currently active font for the *gopen*ed device. All alpha output done by *fm_write* after this point will use this font.

fm_load activates its font, so calling *fm_activate* is often not needed.

HARDWARE DEPENDENCIES

Series 500:

Font manager routines do not support HP-15 (2-byte) fonts on Series 500.

SEE ALSO

fm_load(3W),fm_write(3W).

DIAGNOSTICS

A -1 is returned if *gfd* or *fontid* is not valid; otherwise, 0 is returned. See *errno*(2) for further information.

NAME
fm_clipflag – set clipping flag

SYNOPSIS
int fm_clipflag(gfd,flag);
int gfd;
int flag;

DESCRIPTION
gfd	is an integer file descriptor for an *gopen*ed device interface.
flag	flag to indicate whether clipping is on (1) or off (0).

DISCUSSION
This routine enables clipping for raster alpha output as indicated by *flag*.

HARDWARE DEPENDENCIES
Series 500:

Font manager routines do not support HP-15 (2-byte) fonts on Series 500.

DEFAULTS
Clipping flag is off (0).

SEE ALSO
fm_cliplim(3W).

DIAGNOSTICS
A -1 is returned if *gfd* is not valid. If clipping is not enabled, it is possible to write outside of the window boundaries, randomly change other user variables, destroy the executing program itself, etc. It is recommended that clipping always be enabled for debugging. See *errno*(2) for more information.

NAME

fm_cliplim – set clip limits

SYNOPSIS

int fm_cliplim (gfd,x,y,width,height);
int gfd;
int x,y;
int width,height;

DESCRIPTION

gfd	is an integer file descriptor for an *gopen*ed device interface.
x,y	are the minimum (upper,left) x,y clip boundary, in pixels, relative to the window described by *gfd*.
width,height	are the size, in pixels, of the rectangle to clip.

DISCUSSION

This routine sets the clipping rectangle for raster alpha output as indicated by *x*, *y*, *width*, and *height*.

HARDWARE DEPENDENCIES

Series 500:

Font manager routines do not support HP-15 (2-byte) fonts on Series 500.

SEE ALSO

fm_clipflag(3W).

DIAGNOSTICS

Returns -1 if *gfd* is not valid; otherwise, 0 is returned. See *errno*(2) for more information.

NAME

fm_colors – set active font's foreground and background colors

SYNOPSIS

int fm_colors(gfd,foreground,background);
int gfd,foreground,background;

DESCRIPTION

gfd	is an integer file descriptor for an *gopen*ed device interface.
foreground	is the new foreground color; valid values are device dependent.
background	is the new background color; valid values are device dependent.

DISCUSSION

Sets the foreground and background colors as would be used by *fm_write.*

Though the colors are device dependent, it is generally safe to assume that 0 is black (off) and 1 is white (on).

HARDWARE DEPENDENCIES

Series 500:

Font manager routines do not support HP-15 (2-byte) fonts on Series 500.

SEE ALSO

fm_write(3w).

DIAGNOSTICS

Returns -1 if *gfd* is not valid or the color is out of range. See *errno*(2) for more information.

NAME

fm_fileinfo – return the size of cells in a font file

SYNOPSIS

int fm_fileinfo(path,width,height,baseline);
char *path;
int *width,*height,*baseline;

DESCRIPTION

path	is the path name to the font file.
width	width of 1-byte character cells in the font.
height	height of character cells in the font.
baseline	baseline of character cells in the font; distance from the bottom of the character cell to the bottom of a typical character image which does not have a descender.

DISCUSSION

For the designated font file, the width, height and baseline for the character cells are returned. The returned cell width is the cell width of the 1-byte characters, even if the designated font is a HP-15 (2-byte) character font that includes 1- and 2-byte characters. 2-byte characters are twice as wide as the returned width. The baseline is also returned for the 1-byte character cell, regardless of whether a character is one or two bytes, and the baseline of the 2-byte characters is equal to the descender line of 1/2-byte characters.

For proportional fonts, width and height are the maximums, respectively, of the widths and heights of the character cells in the font file. The baseline is sometimes useful for lining up characters in different-sized fonts.

This library call is useful for determining what size to create a window that will contain fixed text.

HARDWARE DEPENDENCIES

Series 500:

Font manager routines do not support HP-15 (2-byte) fonts on Series 500.

SEE ALSO

fm_rasterinfo(3W),fm_load(3W).

DIAGNOSTICS

A -1 is returned if *path* does not designate a font file; otherwise, 0 is returned. See *errno*(2) for more information.

NAME

fm_fontdir – set character direction

SYNOPSIS

int fm_fontdir(gfd,direction);
int gfd, direction;

DESCRIPTION

gfd	is an integer file descriptor for an *gopen*ed device interface.
direction	Direction for character printing, valid values are:

'u'	-	write characters upward
'd'	-	write characters downward
'l'	-	write characters to the left
'r'	-	write characters to the right

DISCUSSION

Sets the character printing direction as defined by *direction*. This write direction remains in effect until a different font is activated.

HARDWARE DEPENDENCIES

Series 500:

Font manager routines do not support HP-15 (2-byte) fonts on Series 500.

SEE ALSO

fm_write(3W).

DIAGNOSTICS

Returns -1 if *gfd* is not valid or *direction* is out of range; otherwise, 0 is returned. See *errno*(2) for more information.

NAME

fm_getname – translate font id to filename

SYNOPSIS

int fm_getname(gfd,fontid,filename);
int gfd,fontid;
char *filename;

DESCRIPTION

gfd	is an integer file descriptor for an *gopen*ed device interface.
fontid	is the font id as returned by *fm_load.*
filename	array of characters to contain font filename.

DISCUSSION

The filename that was given to *fm_load* to create *fontid* is returned in *filename.*

It is the responsibility of the caller to assure that *filename* points to a area large enough to contain the entire filename.

HARDWARE DEPENDENCIES

Series 500:

Font manager routines do not support HP-15 (2-byte) fonts on Series 500.

SEE ALSO

fm_load(3W).

DIAGNOSTICS

A -1 is returned if *gfd* or *fontid* is not valid; otherwise, 0 is returned. See *errno*(2) for more information.

NAME

fm_load – load a font into memory

SYNOPSIS

int fm_load(gfd,path,fontid);
int gfd;
char *path;
int *fontid;

DESCRIPTION

gfd	is an integer file descriptor for an *gopen*ed device interface.
path	is the path name of the font file to be loaded.
fontid	is the system wide font id returned upon successful load.

DISCUSSION

This routine loads a font file and updates the font manager's tables appropriately. The designated font is made the active font, as if *fm_activate* was called. Hence, certain graphics parameters may be reset; see *fm_activate* for details. If the optimize bit is set in the font file *fm_opt* will be called to optimize the font. A system-wide unique font identifier is returned.

Note that font ids (and hence fonts) are associated with a (gfd,process-id) pair. In particular, if a font has been loaded using a gfd for one device, *fm_write* cannot be called with that font id and a different gfd. Both the file descriptor and process id must match to use a font.

Note that *fm_load* and *fm_remove* act much like *open*(2) and *close*(2), in that every font loaded by *fm_load* should be released by a call to *fm_remove.* If a font is loaded twice by separate calls to *fm_load*, two calls to *fm_remove* should be performed to release the font. If *gfd* or *path* is invalid, -1 is returned by *fm_load.* If the font is already loaded a new *fontid* will be returned. A -1 is also returned if format of the font file is incompatible with the device architecture.

HARDWARE DEPENDENCIES

Series 500:

Font manager routines do not support HP-15 (2-byte) fonts on Series 500.

SEE ALSO

fm_remove(3W),fm_activate(3W),fm_opt(3W).

DIAGNOSTICS

A return of -1 indicates failure; otherwise, 0 is returned. See *errno*(2) for more information.

WARNING

If a rectangular font (pixelformat = 1) or a HP-15 (2-byte) font (pixelformat = 2) is used on a low resolution display with retained rasters, every other rectangular pixel will be lost when an area is obscured. Also 2-byte characters that are written to an obscured area will be twice as big as they should be when the area is unobscured.

NAME

fm_opt – optimize character generation if possible

SYNOPSIS

int fm_opt(gfd,optmode);
int gfd, optmode;

DESCRIPTION

gfd	is an integer file descriptor for an *gopen*ed device interface.
optmode	optimization mode, where 0 means unoptimize and 1 means attempt to optimize.

DISCUSSION

This routine causes the device driver to use any special hardware and/or routines it has available for displaying characters for the active font. Not all fonts are optimizable. This is completely device dependent; some devices may not support such a thing as optimization so all character generation may be the same speed regardless.

If optimization is unsuccessful, it may be due to lack of internal table space. It may help to release an optimized font and try again.

If the optimize bit is set in the font file, *fm_load* will call *fm_opt*, thereby attempting to optimize the font as it is loaded.

Only the 1-byte character portion of a HP-15 (2-byte) font will be optimized. The 2-byte character portion will remain bit/pixel in memory and will continue to be expanded to byte/pixel at display time.

HARDWARE DEPENDENCIES

Series 500:

Font manager routines do not support HP-15 (2-byte) fonts on Series 500.

SEE ALSO

fm_load(3W).

DIAGNOSTICS

This routine will return -1 if the operation was unsuccessful (or not possible at this time due to limited resources) or if there is no active font; otherwise, 0 is returned. See *errno*(2) for more information.

NAME

fm_rasterinfo – return the size of cells in a font

SYNOPSIS

int fm_rasterinfo(gfd,fontid,width,height,baseline);
int gfd,fontid;
int *width,*height,*baseline;

DESCRIPTION

gfd	is an integer file descriptor for an *gopen*ed device interface.
fontid	font id as returned by *fm_load.*
width	width of 1-byte character cells in the font.
height	height of character cells in the font.
baseline	baseline of character cells in the font.

DISCUSSION

For the designated font, the width, height and baseline for the character cells are returned. The returned cell width is the width of 1-byte characters, even if the designated font is a HP-15 (2-byte) font that includes both 1- and 2-byte characters. The 2-byte characters are twice as wide as the 1-byte characters. The returned value for baseline is for the 1-byte character cell. The baseline of the 2-byte characters is equal to the descender line of 1/2-byte characters.

This library call is useful for determining what size to create a window that will contain fixed text.

For proportional fonts, width and height are the maximums, respectively, of the widths and heights of the character cells in the font. The baseline is sometimes useful for lining up characters in different-sized fonts.

HARDWARE DEPENDENCIES

Series 500:

Font manager routines do not support HP-15 (2-byte) fonts on Series 500.

SEE ALSO

The "Font Manager" chapter of the *Fast Alpha/Font Manager Programmer's Manual* and the following reference pages: fm_fileinfo(3W),fm_load(3W).

DIAGNOSTICS

A -1 is returned if *gfd* or *fontid* is not valid; otherwise, 0 is returned. See *errno*(2) for more information.

NAME

fm_remove – remove a font

SYNOPSIS

int fm_remove(gfd,fontid);
int gfd,fontid;

DESCRIPTION

gfd	is an integer file descriptor for an *gopen*ed device interface.
fontid	is the id of font to be removed.

DISCUSSION

This routine deletes the font and makes it unavailable for further use by the user. All fonts should be removed via *fm_remove* calls before *gclose* is called.

HARDWARE DEPENDENCIES

Series 500:

Font manager routines do not support HP-15 (2-byte) fonts on Series 500.

SEE ALSO

fm_load(3W).

DIAGNOSTICS

A -1 is returned if *gfd* is invalid or *fontid* is not valid for this process/window; otherwise, 0 is returned. See *errno*(2) for more information.

NAME

fm_sixteen_bit – set sixteen bit flag

SYNOPSIS

int fm_sixteen_bit(gfd,flag);
int gfd;
int flag;

DESCRIPTION

gfd	is an integer file desciptor for an gopened device interface.
flag	flag to indicate whether sixteen bit mode is on (1) or off (0).

DISCUSSION

This routine enables sixteen bit mode for raster alph output as indicated by flag.

HARDWARE DEPENDENCIES

Series 500:

Font manager routines do not support MIT X11 16 bit fonts on Series 500.

DEFAULTS

Sixteen bit mode is off (0).

SEE ALSO

XDrawString(3X).

DIAGNOSTICS

Sixteen bit mode allows for standard MIT X11 16 bit raster fonts to be displayed. MIT X11 16 bit raster fonts are not HP-15 fonts.

NAME

fm_str_len – determine the pixel length of a character string

SYNOPSIS

int fm_str_len(gfd,str,strlen);
int gfd;
char *str;
int strlen;

DESCRIPTION

gfd	is an integer file descriptor for a *gopen*ed device interface.
str	is the string for which the length is to be determined.
strlen	is the length of *str* in bytes.

DISCUSSION

The return value of this function is the length, in pixels, of a character string along the current character direction of the active font for this process. For example, this could be used to center text in a proportionately spaced font.

Note that *str* is not null-terminated; instead, *strlen* is used to determine its length. Each 2-byte character is treated as two bytes.

The following processing is done only when the active font is a HP-15 (2-byte) character font:

* If the string contains an undefined 2-byte character, the cell size of the 2-byte galley character is used.
* If the string contains an illegal 2-byte character, the illegal character is treated as the two 1-byte characters that correspond to the two codes that compose the illegal 2-byte character code.
* If the last code of the string is the first code of a 2-byte character, the character is treated as the 1-byte character that corresponds to the code value.

HARDWARE DEPENDENCIES

Series 500:

Font manager routines do not support HP-15 (2-byte) fonts on Series 500.

SEE ALSO

fm_write(3W).

DIAGNOSTICS

A -1 is returned if *gfd* is not valid; otherwise, 0 is returned. See *errno*(2) for more information.

NAME

fm_styleinto – return style information about a font

SYNOPSIS

#include <fonticon.h>
int fm_styleinfo(gfd,fontid,symbol_char,escapecodes);
int gfd;
int fontid;
char *symbol_char;
struct escapecodes *escapecodes;

DESCRIPTION

gfd	is an integer file descriptor for an *gopen*ed device interface.
fontid	is the font id as returned by *fm_load.*
symbol_char	character describing font (e.g. 'U').
escapecodes	is a structure containing font information, see **/usr/include/fonticon.h**.

DISCUSSION

A font is sometimes designated by the (symbol_int, symbol_char) pair, especially in escape sequences (see also term0 documentation). For example, 8U is an eight-bit Roman font, 0L is a line drawing font, 0M is a math symbol font, etc.

HARDWARE DEPENDENCIES

Series 500:

Font manager routines do not support HP-15 (2-byte) fonts on Series 500.

SEE ALSO

fm_rasterinfo(3W),fm_load(3W).

DIAGNOSTICS

A -1 is returned if *fontid* or *gfd* is not valid; otherwise, 0 is returned. See *errno*(2) for more information.

NAME

fm_write – write characters to the screen

SYNOPSIS

int fm_write(gfd,x,y,str,strlen,dump,colormode);
int gfd,x,y;
char *str;
int strlen,dump,colormode;

DESCRIPTION

gfd	is an integer file descriptor for an *gopen*ed device interface.
x,y	location (upper left) to begin writing characters.
str	character string to be output.
strlen	is the length of *str* in bytes.
dump	is a boolean indicator: update the screen immediately (TRUE), let the system imposed buffering take care of the visual update (FALSE).
colormode	is a boolean indicator: use the colors from *fm_colors* (TRUE).

DISCUSSION

If *colormode* is not enabled, the string is written using the current values of drawing mode and write enable mask for file descriptor *gfd*; the value of the background color index has no effect. Neither the drawing mode, the write enable mask, nor the background color index is modified by *fm_write*.

If *colormode* is enabled, the string is written with the foreground and background colors established by *fm_colors*. Neither the drawing mode, the write enable mask, nor the background color index has any effect on the result. Neither the drawing mode nor the background color index is modified by *fm_write*. A side effect of *fm_write* (in colormode) is that all planes are enabled for writing.

The string *str* is not null-terminated, the parameter *strlen* determines how many bytes are written.

The following processing is done only when the current active font is a HP-15 (2-byte) character font:

* If the string to be output contains an undefined 2-byte character, the 2-byte galley character is output instead of the undefined character.
* If the string contains an illegal 2-byte character, the two 1-byte characters that correspond to the codes of the illegal 2-byte character are output.
* If the last code of the string is the first code of a 2-byte character, the 1-byte character that corresponds to the code value is output.

WARNINGS

On the Series 500, and prior to release 5.16 on the Series 200/300, if *colormode* is enabled, both the background and foreground color index and the drawing mode will be modified by *fm_write*.

HARDWARE DEPENDENCIES

Series 500:

Font manager routines do not support HP-15 (2-byte) fonts on Series 500.

SEE ALSO

fm_colors(3W),fm_load(3W),write_enable(3G),drawing_mode(3G), background_color_index(3G).

DIAGNOSTICS

A -1 is returned if *gfd* is not valid; otherwise, 0 is returned. See *errno*(2) for more information.

BUGS

If a rectangular font (pixelformat = 1) or a HP-15 (2-byte) font (pixelformat = 2) is used on a low resolution display with retained rasters, every other rectangular pixel will be lost when an area is obscured. Also, 2-byte characters that are written to the obscured area will be twice as big as they should be when the area is unobscured.

NAME

fontm – summary of font manager library routines

DISCUSSION

The font manager library, **/usr/lib/libfontm.a**, provides a high-performance alpha (textual) interface to graphics windows and bitmapped displays. The font manager can handle different sizes of fonts and provides an x,y pixel interface (as opposed to the row and column interface of fast alpha).

All font manager library calls require a file descriptor from *gopen(3G)* which may be that of a bitmapped display device or of a graphics window.

Programs that call font manager routines must link in the font manager (-lfontm) and Starbase (-lsb1 -lsb2) libraries.

The header file **/usr/include/fonticon.h** contains type and constant definitions used by font manager routines. Programs should use these definitions when calling font manager library routines.

Font manager library routines are summarized below. For more information on each routine, consult its reference page.

fm_activate(3W)	Make a loaded font the *active* font.
fm_clipflag(3W)	Set clipping flag. This enables or disables the ability to write outside a window's boundaries.
fm_cliplim(3W)	Set the clip limits; that is, define the area of a window in which clipping pertains.
fm_colors(3W)	Set the active font's foreground and background colors.
fm_fileinfo(3W)	Given the path name of a font file, this routine returns size information (i.e., pixel width, height, and baseline) about the font's character cells.
fm_fontdir(3W)	Set the direction for writing characters.
fm_getname(3W)	Return the path name of the font's definition file.
fm_load(3W)	Load a font file into memory and activate it.
fm_opt(3W)	Optimize character generation if possible.
fm_rasterinfo(3W)	This routine returns size information (i.e., pixel width, height, and baseline) about the font's character cells.
fm_remove(3W)	Remove a font from memory. Once a font is removed from memory, it must be reloaded, via *fm_load*(3W), before it can be used again.
fm_str_len(3W)	Return the pixel length of a character string. The length is determined from the font size of the active font.
fm_styleinfo(3W)	Return style information about a font. This information is returned in an *escapecodes* structure, as defined in **/usr/include/fonticon.h**.
fm_write(3W)	Use the active font to display characters.

EXAMPLES

The following example compiles a program, named *fontmprog.c*, that calls font manager routines to display text in graphics windows. The program will run on a Series 300 low-resolution display using retained windows (thus the -ldd300l and -lddbyte options).

```
cc fontmprog.c -lfontm -ldd300l -lddbyte -lwindow -lsb1 -lsb2
```

SEE ALSO

windows(1),fa(3W),gopen(3G),window(3W).

COMMAND/KEYWORD INDEX

WHAT IT IS!

This is an "in context" index. It is sometimes called a permuted index. It is generated from the **NAME** part, i.e., the command, or routine name, and its description part, of the reference pages. Each significant word in the command/description line is used as an index entry, a keyword. It is "in context" because the words in the command/description line surrounding, i.e., preceding and/or following, the keyword are included with it. In certain cases, especially when more than one command is included in the **NAME** part of a reference page, some license is used to select command line information.

HOW IT IS!

These conventions apply:

commands

- a command, or routine name, is printed in *italics* and followed by a colon (:)

context

- the context for the keyword is contained in the left and center columns
- to read the context of a keyword, start with the *command* or, if the command part has been truncated, with the right brace (}); read to the end of the center column and wrap around to the beginning of the left column; read across to the *command* or either brace

keyword

- the keyword, or look-up word, for an index entry is the left word in the center column, i.e., the one under the ↓
- keywords are in alphabetical order; uppercase is folded into lowercase
- if a keyword is a command it is preceded by an asterisk (*)

PAGE NAME

- the PAGE NAME, where the command is presented in this reference, appears in the right column; the number and/or letter in parentheses identifies the section where this referenced PAGE NAME is located

special characters

*	an asterisk indicates the keyword is a command
:	a colon separates a command or routine name from its description
{	a left brace indicates where the end of the command line is truncated
}	a right brace indicates where the beginning of the command line is truncated

truncation

- if a command line, including the command and the description, is too long to fit in the context area, the end and/or the beginning of the line is truncated
- braces are used to indicate where a truncation occurs

EXAMPLE

Here is a command/description from this reference manual:

fainit - prepare a fast alpha window device for output, and set up all defaults

And here are the entries produced for the index:

	↓ **command*/keyword	PAGE_NAME
and set{ *fainit*: prepare a fast	alpha window device for output, ...	FAINIT(3W)
for output, and set up all	defaults }alpha window device	FAINIT(3W)
all{ }prepare a fast alpha window	device for output, and set up	FAINIT(3W)
window device for output, and{	**fainit*: prepare a fast alpha	FAINIT(3W)
output, and{ *fainit*: prepare a	fast alpha window device for	FAINIT(3W)
}a fast alpha window device for	output, and set up all defaults	FAINIT(3W)
device for output, and{ *fainit*:	prepare a fast alpha window	FAINIT(3W)
window device for output, and	set up all defaults }fast alpha	FAINIT(3W)
fainit: prepare a fast alpha	window device for output, and{	FAINIT(3W)

	↓ **command*/keyword	PAGE_NAME
fafontactivate:	activate a fast alpha font	FAFONTACTIVATE(3W)
font cache and prepare it for	activation }font into the user's	FAFONTLOAD(3W)
fm_activate: make a font	active	FM_ACTIVATE(3W)
background{ *fm_colors*: set	active font's foreground and	FM_COLORS(3W)
get information about the fast	alpha environment *fagetinfo*:	FAGETINFO(3W)
set information about the fast	alpha environment *fasetinfo*:	FASETINFO(3W)
terminate the current fast	alpha environment *faterminate*:	FATERMINATE(3W)
fafontactivate: activate a fast	alpha font	FAFONTACTIVATE(3W)
facolors: set the fast	alpha font foreground and{	FACOLORS(3W)
fa: summary of fast	alpha library routines	FA(3W)
and set{ *fainit*: prepare a fast	alpha window device for output,	FAINIT(3W)
specified{ *farectwrite*: fill an	area of the window with the	FARECTWRITE(3W)
faclear: clear the window	area specified by the given{	FACLEAR(3W)
fast alpha font foreground and	background colors }set the	FACOLORS(3W)
set active font's foreground and	background colors *fm_colors*:	FM_COLORS(3W)
}load a font into the user's font	cache and prepare it for{	FAFONTLOAD(3W)
a font from the user's font	cache *fafontremove*: remove	FAFONTREMOVE(3W)
fm_fileinfo: return the size of	cells in a font file	FM_FILEINFO(3W)
return the size of	cells in a font *fm_rasterinfo*:	FM_RASTERINFO(3W)
fm_fontdir: set	character direction	FM_FONTDIR(3W)
of the window with the specified	character }fill an area	FARECTWRITE(3W)
fm_opt: optimize	character generation if possible	FM_OPT(3W)
determine the pixel length of a	character string *fm_str_len*:	FM_STR_LEN(3W)
fm_write: write	characters to the screen	FM_WRITE(3W)
fawrite: write a line of	characters with their{	FAWRITE(3W)
by the given rectangle *faclear*:	clear the window area specified	FACLEAR(3W)
fm_cliplim: set	clip limits	FM_CLIPLIM(3W)
fm_clipflag: set	clipping flag	FM_CLIPFLAG(3W)
font foreground and background	colors }set the fast alpha	FACOLORS(3W)
font's foreground and background	colors *fm_colors*: set active	FM_COLORS(3W)
facursor:	control the displayed cursor	FACURSOR(3W)
faterminate: terminate the	current fast alpha environment	FATERMINATE(3W)
facursor: control the displayed	cursor	FACURSOR(3W)
for output, and set up all	defaults }alpha window device	FAINIT(3W)
character string *fm_str_len*:	determine the pixel length of a	FM_STR_LEN(3W)
all{ }prepare a fast alpha window	device for output, and set up	FAINIT(3W)
fm_fontdir: set character	direction	FM_FONTDIR(3W)
facursor: control the	displayed cursor	FACURSOR(3W)
a line of characters with their	enhancements *fawrite*: write	FAWRITE(3W)
information about the fast alpha	environment *fagetinfo*: get	FAGETINFO(3W)
information about the fast alpha	environment *fasetinfo*: set	FASETINFO(3W)
terminate the current fast alpha	environment *faterminate*:	FATERMINATE(3W)
library routines	**fa*: summary of fast alpha	FA(3W)
specified by the given{	**faclear*: clear the window area	FACLEAR(3W)
font foreground and background{	**facolors*: set the fast alpha	FACOLORS(3W)
cursor	**facursor*: control the displayed	FACURSOR(3W)
alpha font	**fafontactivate*: activate a fast	FAFONTACTIVATE(3W)
user's font cache and prepare{	**fafontload*: load a font into the	FAFONTLOAD(3W)
the user's font cache	**fafontremove*: remove a font from	FAFONTREMOVE(3W)
the fast alpha environment	**fagetinfo*: get information about	FAGETINFO(3W)

: separates command from description; } indicates location of leading truncation; { indicates location of trailing truncation;

	↓ *command/keyword	PAGE_NAME
window device for output, and{	*fainit: prepare a fast alpha	FAINIT(3W)
window with the specified{	*farectwrite: fill an area of the	FARECTWRITE(3W)
window	*faroll: roll a portion of the	FAROLL(3W)
the fast alpha environment	*fasetinfo: set information about	FASETINFO(3W)
}get information about the	fast alpha environment	FAGETINFO(3W)
}set information about the	fast alpha environment	FASETINFO(3W)
}terminate the current	fast alpha environment	FATERMINATE(3W)
fafontactivate: activate a	fast alpha font	FAFONTACTIVATE(3W)
background{ facolors: set the	fast alpha font foreground and	FACOLORS(3W)
fa: summary of	fast alpha library routines	FA(3W)
output, and{ fainit: prepare a	fast alpha window device for	FAINIT(3W)
current fast alpha environment	*faterminate: terminate the	FATERMINATE(3W)
characters with their{	*fawrite: write a line of	FAWRITE(3W)
the size of cells in a font	file fm_fileinfo: return	FM_FILEINFO(3W)
fm_getname: translate font id to	filename	FM_GETNAME(3W)
the specified{ farectwrite:	fill an area of the window with	FARECTWRITE(3W)
fm_clipflag: set clipping	flag	FM_CLIPFLAG(3W)
	*fm_activate: make a font active	FM_ACTIVATE(3W)
	*fm_clipflag: set clipping flag	FM_CLIPFLAG(3W)
	*fm_cliplim: set clip limits	FM_CLIPLIM(3W)
foreground and background{	*fm_colors: set active font's	FM_COLORS(3W)
cells in a font file	*fm_fileinfo: return the size of	FM_FILEINFO(3W)
direction	*fm_fontdir: set character	FM_FONTDIR(3W)
filename	*fm_getname: translate font id to	FM_GETNAME(3W)
	*fm_load: load a font into memory	FM_LOAD(3W)
generation if possible	*fm_opt: optimize character	FM_OPT(3W)
of cells in a font	*fm_rasterinfo: return the size	FM_RASTERINFO(3W)
	*fm_remove: remove a font	FM_REMOVE(3W)
	*fm_sixteen_bit: set 16 bit flag	FM_SIXTEEN_BIT(3W)
length of a character string	*fm_str_len: determine the pixel	FM_STR_LEN(3W)
information about a font	*fm_styleinto: return style	FM_STYLEINFO(3W)
the screen	*fm_write: write characters to	FM_WRITE(3W)
fm_activate: make a	font active	FM_ACTIVATE(3W)
}load a font into the user's	font cache and prepare it for{	FAFONTLOAD(3W)
remove a font from the user's	font cache fafontremove:	FAFONTREMOVE(3W)
activate a fast alpha	font fafontactivate:	FAFONTACTIVATE(3W)
return the size of cells in a	font file fm_fileinfo:	FM_FILEINFO(3W)
return the size of cells in a	font fm_rasterinfo:	FM_RASTERINFO(3W)
fm_remove: remove a	font	FM_REMOVE(3W)
return style information about a	font fm_styleinto:	FM_STYLEINFO(3W)
facolors: set the fast alpha	font foreground and background{	FACOLORS(3W)
fafontremove: remove a	font from the user's font cache	FAFONTREMOVE(3W)
fm_getname: translate	font id to filename	FM_GETNAME(3W)
fm_load: load a	font into memory	FM_LOAD(3W)
and prepare{ fafontload: load a	font into the user's font cache	FAFONTLOAD(3W)
fontm: summary of	font manager library routines	FONTM(3W)
library routines	*fontm: summary of font manager	FONTM(3W)
colors fm_colors: set active	font's foreground and background	FM_COLORS(3W)
}set the fast alpha font	foreground and background colors	FACOLORS(3W)
fm_colors: set active font's	foreground and background colors	FM_COLORS(3W)

: separates command from description; } indicates location of leading truncation; { indicates location of trailing truncation;

fm_opt: optimize character	generation if possible	FM_OPT(3W)
	↓ **command*/keyword	PAGE_NAME
the window area specified by the	given rectangle *faclear*: clear	FACLEAR(3W)
fm_getname: translate font	id to filename	FM_GETNAME(3W)
fm_styleinfo: return style	information about a font	FM_STYLEINFO(3W)
environment *fagetinfo*: get	information about the fast alpha	FAGETINFO(3W)
environment *fasetinfo*: set	information about the fast alpha	FASETINFO(3W)
fm_str_len: determine the pixel	length of a character string	FM_STR_LEN(3W)
fa: summary of fast alpha	library routines	FA(3W)
fontm: summary of font manager	library routines	FONTM(3W)
fm_cliplim: set clip	limits	FM_CLIPLIM(3W)
enhancements *fawrite*: write a	line of characters with their	FAWRITE(3W)
fm_load:	load a font into memory	FM_LOAD(3W)
cache and prepare{ *fafontload*:	load a font into the user's font	FAFONTLOAD(3W)
fontm: summary of font	manager library routines	FONTM(3W)
fm_load: load a font into	memory	FM_LOAD(3W)
possible *fm_opt*:	optimize character generation if	FM_OPT(3W)
}a fast alpha window device for	output, and set up all defaults	FAINIT(3W)
fm_str_len: determine the	pixel length of a character{	FM_STR_LEN(3W)
faroll: roll a	portion of the window	FAROLL(3W)
optimize character generation if	possible *fm_opt*:	FM_OPT(3W)
device for output, and{ *fainit*:	prepare a fast alpha window	FAINIT(3W)
into the user's font cache and	prepare it for activation }font	FAFONTLOAD(3W)
area specified by the given	rectangle }clear the window	FACLEAR(3W)
fm_remove:	remove a font	FM_REMOVE(3W)
font cache *fafontremove*:	remove a font from the user's	FAFONTREMOVE(3W)
font *fm_styleinfo*:	return style information about a	FM_STYLEINFO(3W)
font file *fm_fileinfo*:	return the size of cells in a	FM_FILEINFO(3W)
font *fm_rasterinfo*:	return the size of cells in a	FM_RASTERINFO(3W)
faroll:	roll a portion of the window	FAROLL(3W)
summary of fast alpha library	routines *fa*:	FA(3W)
summary of font manager library	routines *fontm*:	FONTM(3W)
write characters to the	screen *fm_write*:	FM_WRITE(3W)
background colors *fm_colors*:	set active font's foreground and	FM_COLORS(3W)
fm_fontdir:	set character direction	FM_FONTDIR(3W)
fm_cliplim:	set clip limits	FM_CLIPLIM(3W)
fm_clipflag:	set clipping flag	FM_CLIPFLAG(3W)
alpha environment *fasetinfo*:	set information about the fast	FASETINFO(3W)
foreground and{ *facolors*:	set the fast alpha font	FACOLORS(3W)
window device for output, and	set up all defaults }fast alpha	FAINIT(3W)
fm_fileinfo: return the	size of cells in a font file	FM_FILEINFO(3W)
fm_rasterinfo: return the	size of cells in a font	FM_RASTERINFO(3W)
faclear: clear the window area	specified by the given rectangle	FACLEAR(3W)
an area of the window with the	specified character }fill	FARECTWRITE(3W)
the pixel length of a character	string *fm_str_len*: determine	FM_STR_LEN(3W)
fm_styleinfo: return	style information about a font	FM_STYLEINFO(3W)
routines *fa*:	summary of fast alpha library	FA(3W)
routines *fontm*:	summary of font manager library	FONTM(3W)
environment *faterminate*:	terminate the current fast alpha	FATERMINATE(3W)
fm_getname:	translate font id to filename	FM_GETNAME(3W)
fafontload: load a font into the	user's font cache and prepare it{	FAFONTLOAD(3W)

: separates command from description; **}** indicates location of leading truncation; **{** indicates location of trailing truncation;

remove a font from the	user's font cache *fafontremove*:	FAFONTREMOVE(3W)
	↓ ∗*command*/keyword	PAGE_NAME
given{ *faclear*: clear the	window area specified by the	FACLEAR(3W)
fainit: prepare a fast alpha	window device for output, and{	FAINIT(3W)
faroll: roll a portion of the	window	FAROLL(3W)
farectwrite: fill an area of the	window with the specified{	FARECTWRITE(3W)
their enhancements *fawrite*:	write a line of characters with	FAWRITE(3W)
fm_write:	write characters to the screen	FM_WRITE(3W)

: separates command from description; **}** indicates location of leading truncation; **{** indicates location of trailing truncation;

Font Information

Getting Font Information

A program is provided with the Fast Alpha/Font Managers libraries that will enable you to extract information concerning the raster fonts. The following steps will allow you to compile and execute the `fontinfo` program. The resultant display will give you font information.

1. Log in as superuser.
2. Type `cd` to `/usr/lib/fa_fm_demo`
3. Type `make -f Makefile fontinfo`
4. Type `fontinfo /usr/lib/raster/`⟨*font directory*⟩`/`⟨*font name*⟩
   (See table A-1 for font directories and names.)

   For example:

   ```
   fontinfo /usr/lib/raster/8x16/lp.8u
   ```

When you execute `fontinfo` on a Hewlett-Packard raster font, a summary of font information is printed to the standard output.

Raster Fonts

This is a list of the fonts provided with the Fast Alpha/Font Manager libraries. For each font size, the corresponding HP-UX directory path is shown.

Table A-1. Font Directories and Names

Font Directory	Font Name	Description

`/usr/lib/raster/10x20`	`kana.8K`
	`lp.8U`
	`lp.b.8U`
		Katakana, 8 bit
		Roman8, ascii, 8 bit
		Roman8, ascii, 8 bit (bold)

`/usr/lib/raster/12x20`	`cour.0` `Ucour.b.0U`	Courier style, ascii, 8 bit Courier style, ascii, 8 bit (bold)
`/usr/lib/raster/12x24`	`kana.8K` `kanji.16K`	Katakana (Japanese), 8 bit Kanji (Japanese), 16 bit
`/usr/lib/raster/18x30`	`math.0M` `pica.8U`	Greek letters, 8 bit Pica style, ascii, 8 bit
`/usr/lib/raster/5x18`	`kana.8K` `kanji.16K`	Katakana (Japanese), 8 bit Kanji (Japanese), 16 bit
`/usr/lib/raster/6x24`	`kana.8K` `kanji.16K`	Katakana (Japanese), 8 bit Kanji (Japanese), 16 bit

`/usr/lib/raster/6x8`	`lp.8U`
	`lp.b.8I`
	`math.8M`
		Roman8, ascii, 8 bit
		Roman8, ascii, 8 bit (bold)
		Greek letters, 8 bit

`/usr/lib/raster/7x10`	`lp.8U`	Roman8, ascii, 8 bit

`/usr/lib/raster/8x16`	`kana.8K`
	`linedraw.0L`
	`lp.8U`
	`lp.b.8U`
	`lp.i.8U`
	`math.0M`
		Katakana (Japanese), 8 bit
		Line drawing characters, 8 bit
		Roman8, ascii, 8 bit
		Roman8, ascii, 8 bit (bold)
		Roman8, ascii, 8 bit (italic)
		Greek letters, 8 bit

Index

A

activating a fast alpha font 2-19
activating a font manager font 3-5
active font, fast alpha 2-18
active font, font manager 3-3
alternate font file, term0 4-3

B

`backgroundplanes`, of the fast alpha `fainfo` structure 2-8, 2-10
base font file, term0 4-3
baseline height of font 3-17
`boldness`, of font manager `escapecodes` structure 3-20

C

cache, font manger 3-3
cache for fast alpha fonts 2-18
`capabilities`, of fast alpha `fainfo` structure 2-8, 2-10
changing the fast alpha environment 2-8
character clipping, font manager 3-14
character enhancements, fast alpha 2-4
`clearbeforewrite`, of fast alpha `fainfo` structure 2-8, 2-11
`clearbeforewrite`, of the fast alpha `fainfo` structure 2-8, 2-10
`clearbeforewrite`, relation to font manager writes 3-7
Clear.c program 2-22
clearing a fast alpha rectangle 2-21
clip limits, font manager 3-14
clip limits, setting for font manager 3-15
clipping disabled, font manager 3-14
clipping enabled, font manager 3-14
clipping, font manager 3-14
color, setting for fast alpha fonts 2-19
color with font manager 3-7

colormode, of fast alhpa fainfo structure 2-11
colormode, of fast alpha fainfo structure 2-8
colormode, of the fast alpha fainfo structure 2-8, 2-10
colors, font manager 3-4
compiling programs 2-6, 4-2
compiling window programs 4-1
controlling direction of font manager writes 3-9
cursor control, fast alpha 2-13
cursor, of the fast alpha fainfo structure 2-8, 2-10
Cursor.c program 2-14

D

defaultenhancements, of fast alpha fainfo structure 2-8, 2-11
defaultenhancements, of the fast alpha fainfo structure 2-8, 2-10
determining string length, font manager 3-12
device driver for CRT, linking with programs 4-2
direction of font manager writes, controlling 3-9
disabling clipping, font manager 3-14

E

enabling clipping, font manager 3-14
enhancements, of the fast alpha fainfo structure 2-8, 2-10
enhancements to fast alpha characters 2-4
environment variables
  LANG 4-3
  WMBASEFONT 2-18
  WMFONTDIR 3-3
escapecodes structure 3-20
example
  FM_clipping.c 3-15
  FM_Colors.c 3-11
  FM_Direction.c 3-10
  FM_Struct.c 3-6
  Font_size.c 3-19
  Style.c 3-21
examples
  changing the fast alpha environment 2-12
  Clear.c 2-22
  clear_gr.c 2-22
  colors, setting for fast alpha font 2-20
  disabling display of fast alpha cursor 2-14

fast alpha cursor, disabling display of 2-14
fast alpha cursor positioning 2-14
fast alpha environment, changing 2-12
fast alpha, loading and activating a font 2-20
fast alpha rectangle, scrolling 2-23
fast alpha, writing characters 2-16
FM_length.c 3-13
font colors, setting for fast alpha 2-20
font manager, getting font information 3-21
font manager, loading a font 3-11
getting font information 3-21
initializing the fast alpha environment 2-7
loading a font manager font 3-11
loading and activating a fast alpha font 2-20
positioning the fast alpha cursor 2-14
scrolling a fast alpha rectangle 2-23
setting fast alpha font colors 2-20
terminating the fast alpha environment 2-7
writing fast alpha characters 2-16

F

FABONW 2-8, 2-10
faclear 2-21
FACOLOR 2-8, 2-10–11, 2-20
fa_coordinate structure 2-4
facursor 2-13
FACURSORNOMOVE 2-13
fafontload 2-19
fagetinfo 2-8
fa.h header file 2-2, 4-3
Fainfo program 2-12
fainfo structure 2-8
fainfo structure, description of fields 2-8, 2-10
fainit 2-6
FAOFF 2-11, 2-21
fa_rectangle structure 2-4
farectwrite 2-16
faroll 2-23
FAROLLDOWN 2-23
FAROLLLEFT 2-23
FAROLLRIGHT 2-23

FAROLLUP 2-23
fasetinfo 2-8
fast alpha active font 2-18
fast alpha character enhancements 2-4
fast alpha character strings, writing 2-15
fast alpha cursor control 2-13
fast alpha environment, changing 2-8
fast alpha environment, getting information about 2-8
fast alpha environment, initializing 2-3, 2-6
fast alpha environment, setting 2-8
fast alpha environment, terminating 2-3, 2-6
fast alpha font, activating 2-19
fast alpha font cache 2-18
fast alpha font colors, setting 2-19
fast alpha font id 2-18
fast alpha font management 2-18
fast alpha font, removing from cache 2-18
fast alpha fonts, loading 2-19
Fast Alpha Library 2-1
Fast Alpha Library, linking with programs 4-2
Fast Alpha Library, relation to Font Manager Library 3-1
fast alpha, loading fonts 2-18
fast alpha protocol 2-2
fast alpha rectangle, clearing 2-21
fast alpha rectangle, filling with characters 2-16
fast alpha rectangle, scrolling 2-23
fast alpha rectangles 2-4
fast alpha, writing characgters 2-15
Fast Alpha/Font Manager differences 3-1
faterminate 2-6
FAWINDOW 2-6
FAWONB 2-8, 2-10
fawrite 2-15
filling a fast alpha rectangle with characters 2-16
fm_activate 3-5
fm_clipflag 3-14
fm_cliplim 3-15
FM_clipping.c example 3-15
FM_Colors.c example 3-11
FM_Direction.c example 3-10
fm_fileinfo 3-18

fm_fontdir 3-9
fm_getname 3-21
FM_length.c example 3-13
fm_load 3-4
fm_opt 3-12
fm_rasterinfo 3-18
fm_remove 3-5
fm_str_len 3-12
fm_str_len, relation to clipping 3-12, 3-14
FM_Struct.c example 3-6
fm_write 3-7, 3-9
font, activating fast alpha 2-19
font, baseline height 3-17
font cache, font manager 3-3
font cell 3-17
font color, font manager 3-4
font colors, setting for fast alpha 2-19
font directories 4-3
font file 3-3
font files 4-2
font height 3-17
font id, fast alpha 2-18
font id, font manager 3-3
font management, fast alpha 2-18
font management, font manager model 3-3
font manager, activating a font 3-5
font manager, active font 3-3
font manager, clipping 3-14
font manager, color 3-7
font manager, direction of write 3-9
font manager, disabling clipping 3-14
font manager, enabling clipping 3-14
font manager font cache 3-3
font manager, font color 3-4
font manager, font id 3-3
font manager font table 3-3
font manager, getting font size information 3-18
Font Manager Library 3-1
Font Manager Library concepts 3-2
Font Manager Library, linking with programs 4-2
font manager, loading fonts 3-3

font manager, obtaining information about fonts 3-17
font manager, optimizing writes 3-12
font manager, removing a font 3-5
font manager, removing fonts 3-4
font manager, setting clip limits 3-15
font manager, string length 3-12
font manager, write direction control 3-9
font manager, writing characters 3-7
Font Manager/Fast Alpha differences 3-1
font name, getting with font manager 3-21
font size 3-17
font size, getting with font manager 3-18
font style 3-17, 3-20
font style, getting font manager information 3-20
font table, font manager 3-3
font width 3-17
`fontcellheight`, of the fast alpha `fainfo` structure 2-8, 2-10
`fontcellwidth`, of the fast alpha `fainfo` structure 2-8, 2-10
`fonticon.h` header file 3-2, 4-3
Fonts.c program 2-20
`Font_size.c`, example 3-19
`foregroundplanes`, of the fast alpha `fainfo` structure 2-8, 2-10

G

`gclose`(3G) when using fast alpha routines 2-3
getting a font's name with font manager 3-21
getting fast alpha environment information 2-8
getting font manager style information 3-20
getting font size information, font manager 3-18
`gopen`(3G), relation to Font Manager Library 3-2
`gopen`(3G) when using fast alpha routine 2-2
graphics window type, displaying text using Fast Alpha 2-1

H

header files
  `fa.h` 2-2, 4-3
  `fonticon.h` 3-2, 4-3
height of font 3-17
`hpitch`, of font manager `escapecodes` structure 3-20

I

icon definition files 4-3
icon files 4-2
initializing the fast alpha environment 2-6
initializing the fast alpha environment, example 2-7

L

LANG 4-3
`libdddriver.a` 4-2
`libfa.a` 4-2
`libfontm.a` 4-2
libraries, linking with window programs 4-1
`libsb1.a` 4-2
`libsb2.a` 4-2
`libwindow.a` 4-2
link order when compiling window programs 4-2
linking libraries 2-6, 4-2
linking libraries with window programs 4-1
loading fast alpha fonts 2-18
loading font manager fonts 3-4
loading fonts, font manager 3-3

M

`makecurrent`, of fast alpha `fainfo` structure 2-8, 2-11
`makecurrent`, of the fast alpha `fainfo` structure 2-8, 2-10
`MCALWAYS` 2-11

O

optimization of character generation 3-7
optimizing font manager writes 3-12

P

program
  Cursor.c 2-14
  Fainfo 2-12
  Fonts.c 2-20
  Structure 2-7
program Clear.c 2-22
program `FM_clipping.c` 3-15
program `FM_Colors.c` 3-11
program `FM_Direction.c` 3-10

program FM_length.c 3-13
program FM_Struct.c 3-6
program Font_size.c 3-19
program Style.c 3-21
proportional, of font manager escapecodes structure 3-20
protocol, fast alpha 2-2

Q

quality, of font manager escapecodes structure 3-20

R

rectangles, fast alpha 2-4
removing a font manager font 3-5
removing fast alpha fonts 2-18
removing font manager fonts 3-4

S

scrolling a fast alpha rectangle 2-23
setting fast alpha environment information 2-8
setting fast alpha font colors 2-19
setting font manager clip limits 3-15
shared memory, font manager 3-4
size, font 3-17
size, of fast alpha fainfo structure 2-8, 2-10
Starbase Library, linking with programs 4-2
string length, font manager 3-12
Structure program 2-7
style, font 3-17, 3-20
Style.c example 3-21
symbol_int, of font manager escapecodes structure 3-20

T

terminating the fast alpha environment 2-6
terminating the fast alpha environment, example 2-7
text in a graphics window, using Fast Alpha 2-1
text, writing in graphics windows using Font Manager Library 3-1
typeface, of font manager escapecodes structure 3-20

U

/usr/include/fa.h 2-2, 4-3
/usr/include/fonticon.h 3-2, 4-3

/usr/lib/libdddriver.a 4-2
/usr/lib/libfa.a 4-2
/usr/lib/libfontm.a 4-2
/usr/lib/libsb1.a 4-2
/usr/lib/libsb2.a 4-2
/usr/lib/libwindow.a 4-2
/usr/lib/raster/* 4-3
/usr/lib/raster/dflt/a/h/$LANG 4-3
/usr/lib/raster/dflt/a/l/$LANG 4-3
/usr/lib/raster/dflt/b/h/$LANG 4-3
/usr/lib/raster/dflt/b/l/$LANG 4-3
/usr/lib/raster/icons 4-3

V

vheight, of font manager escapecodes structure 3-20

W

width of font 3-17
Window Library, linking with programs 4-2
WMBASEFONT 2-18
WMFONTDIR 3-3
writing characters with font manager 3-7
writing fast alpha character strings 2-15
writing fast alpha characters 2-15
writing font manager characters 3-7

HEWLETT
PACKARD

HP Part Number
98592-90090

Microfiche No. 98592-99090
Printed in U.S.A. E1288

98592-90635

For Internal Use Only