Starbase Device Drivers Library Manual

Volume 2

HP 9000 Series 300/800 Computers

HP Part Number 98592-90018

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

Notices

The information contained in this document is subject to change without notice.

Hewlett-Packard makes no warranty of any kind with regard to this manual, including, but not limited to, the implied warranties of merchantability and fitness for a particular purpose. Hewlett-Packard shall not be liable for errors contained herein or direct, indirect, special, incidental or consequential damages in connection with the furnishing, performance, or use of this material.

Warranty. A copy of the specific warranty terms applicable to your Hewlett-Packard product and replacement parts can be obtained from your local Sales and Service Office.

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.

September 1989 ... Edition 1. This Edition supersedes manual part number 98592-90016.

Contents

PCL

Printer Command Language Formatter

Overview

This section provides a quick overview of the Printer Command Language (PCL) formatter. The PCL formatter is used with both monochromatic and color PCL printers.

In this document, "bitmap" is used to denote a rectangular array of pixels, and can be either a device's frame buffer or an image in memory created by the Starbase Memory Driver. "Starbase bitmap file" is used to denote a bitmap file created by the Starbase procedures `bitmap_to_file` or `dcbitmap_to_file`. The key points are:

1. This formatter permits hard copies from a bitmap or a Starbase bitmap file to a color or monocromatic PCL format printer. The entire bitmap or a subrectangle of the bitmap can be processed and printed. The chapter "Storing Retrieving, and Printing Images" in the *Starbase Graphics Techniques* manual (HP-UX Concepts and Tutorials) should be read prior to reading this document.

2. The following monochromatic PCL printers are supported:
   HP 2225A (ThinkJet)
   HP 2235A (SprintJet)
   HP 2227A and HP2228A (QuietJet and QuietJet Plus)
   HP 2563A, HP2564B, HP2565B, HP2566A, HP2567B
   HP 2686A (LaserJet and LaserJet Plus)
   HP 2932A, HP2933A, HP2934A
   HP 33446A (Laser Jet II)
   HP 33447A (Laser Jet IID)
   C1200A Asian System Printer
   C1202A Asian Serial Printer

3. The following color PCL printers are currently supported:

HP 3630A color (PaintJet)

HP C1602A color (PaintJet XL)

4. Prints can be done in gray scale, monochromatic (black & white), primary (red, green, blue, cyan, yellow, magenta, black, white), or in color.

5. The PCL formatter is not a Starbase driver. In other words, you don't do moves, draws, etc. to the PCL printer. What you can do is:

   - Process and print an already existing image on the bitmap to a color or monochromatic PCL printer with `bitmap_print` or the HP-UX command `screenpr` (see the *Starbase Reference* manual).
   - Process and print an existing bitmap image from a Starbase bitmap file to a color or monochromatic PCL printer with `file_print` or the HP-UX command `pcltrans` (see the *Starbase Reference* manual).

6. The color version of this formatter includes the full monochromatic capabilities. The color version of this formatter works only with HP-UX Release 5.5 and later versions (Series 300), and HP-UX Release 1.2 and later on the Series 800. The monochromatic only version was available with the HP-UX releases 5.2 and 5.3 (Series 300).

7. Graphics prints can be done in 3 ways:

   - Use Starbase procedures to print from a bitmap or Starbase bitmap file under the control of the program which originally creates the bitmap or file.
   - A program other than that which originally creates the bitmap or file can be used in one of two ways.
     a. Use the Starbase procedure gopen without `INIT` followed by the Starbase procedure `bitmap_print` to print a currently displayed bitmap.
     b. Use the Starbase procedure `file_print` to print a previously created Starbase bitmap file.

     HP provides an HP-UX command—`screenpr`—which can be used to print a currently displayed bitmap.
   - Use `pcltrans`, an HP-UX command (see the *Starbase Reference* manual), to spool a Starbase bitmap file to the printer. This is

typically used when the printer is shared, although it can be used on a single-user system to do graphics prints in background.

Printer Configurations

There are two fundamental printer configurations of interest, spooled and non-spooled. The primary difference between the two configurations is that spooling uses the system spooler (`lp`) in the raw (`-oraw` option) mode. This section gives a quick overview of these two configurations so that you can focus in later sections on the information you need for your application.

Non-Spooled Operation

In a non-spooled environment, the following Starbase procedures can be used in your programs:

- `bitmap_print`, `dcbitmap_print`—do a graphics print from the specified bitmap. Remember that the bitmap can either be a display or a memory buffer created by the Starbase memory driver.
- `file_print`—do a graphics print using a file created previously by the `bitmap_to_file` procedure.

Spooled Operation

In a spooled environment, the HP-UX command `pcltrans` (see the *Starbase Reference* manual) is used as a filter to process a Starbase bitmap format file (created previously using `bitmap_to_file`), which is then piped to the `lp` spooler in `raw` mode. Spooling can either be done on a single-user computer or the file can be sent to another computer if the printer is shared.

Alternatively the spooler can be accessed using `bitmap_print` or `file_print`. With these procedures output can be directed to a special device file or redirected through standard out depending upon parameters in the formatter's configuration file. Spooler access can be accomplished by: processing a bitmap or file (using `bitmap_print` or `file_print`) with the output going to standard out. Then redirect or pipe the resulting output to the `lp` spooler in `raw` mode.

Spooler Conflicts

Assume that you have a color or monochromatic PCL printer connected to your system and you (and possibly others) spool graphics prints to it. If you also use Starbase procedures to do graphics prints, the spooler and Starbase program may conflict, producing interleaved/unusable output. Unusable output may occur in both spooled text and the graphics prints. Thus, simultaneous usage of spooled and non-spooled modes should not be used.

If your printer is used for spooling, it is recommended that all graphics prints be done using spooling.

Software Structure

The following files are used for graphics prints on color PCL printers:

```
/usr/lib/starbase/formatters/fmt_table.c
/usr/lib/starbase/formatters/pcl/libfmtpcl.a
/usr/lib/starbase/formatters/pcl/cfg.ctmplt
/usr/bin/pcltrans
/usr/bin/screenpr
```

The following files are used for graphics prints on monochromatic PCL printers:

```
/usr/lib/starbase/formatters/fmt_table.c
/usr/lib/starbase/formatters/pcl/libfmtpcl.a
/usr/lib/starbase/formatters/pcl/cfg.template
/usr/bin/pcltrans
/usr/bin/screenpr
```

Setting Up the Special Device File

A special device file is required if you directly access a printer. If the printer has already been assigned a node as system printer you may use that device file if you are on a single user system and you have write permission for that device file.

If a special device file for your printer has not been assigned, the `mknod` command must be done before proceeding further. You must be superuser to use the `mknod` command. Note that the Select Code is entered in hexadecimal format. For example, if your Select Code is 22, this is entered as 16 hex. The following example sets up a color serial printer (an HP 3630A) on select code 9.

```
* mknod /dev/rp c 1 0x090004
```

You may need to set owner, group, and mode (`chown`,`chgrp`,`chmod`) in an appropriate manner.

When using major number 7 you should ensure that bit 0 of the minor number is set for `raw` mode. Refer to the *HP-UX System Administrator Manual,* "System Administrator's Toolbox" section for further details (in particular the section that covers `mkdev`, and `mknod`).

The Configuration File

The parameters which control printing are specified in two ways:

1. By parameters in the `bitmap_print` and `file_print` procedures. These parameters contain information about the source, e.g., information on the size of the bitmap rectangle to process and print. Parameters are apt to change during program execution. They are discussed in the *Starbase Graphics Techniques* manual and are not discussed here.
2. By additional information contained in files called **configuration files.**

Configuration Files

Configuration files store information about the printer. e.g., resolution, page size, pixel expansion, etc.

Storing printer information in a configuration file is done as a convenience so that you don't need to type in this information each time you use the `bitmap_print` or `file_print` procedures. When you use these procedures, you only need to provide the pathname of the configuration file.

Note All parameters must be present and in the exact order shown. If parameters are missing or incorrect an error will be issued and formatter action will be terminated.

The parameters in the configuration file are :

`ENABLE STANDARD OUT` — If `TRUE`, output goes to standard out regardless of the printer device file specified in the next parameter. If `FALSE`, output goes to the special device file or file specified in the next parameter.

`PRINTER DEVICE FILE` — Specifies the special device file for the printer. This parameter is read but ignored if standard out enable is `TRUE`.

`PRINT METHOD` — Specifies color, color2, primary (red, green, blue, cyan, magenta, yellow, black, white), gray scale, or monochromatic (black and white). This parameter has allowable values of "`color`", "`color2`", "`primary`", "`gray`", "`grey`", "`mono`", and "`monochromatic`".

If `PRINT METHOD` is "`color`", each pixel is converted to an appropriate color. If print method is "`color2`", each pixel is converted to an appropriate color plus a random noise increment value.

If `PRINT METHOD` is "primary" then each pixel is converted to the nearest primary color.

If PRINT METHOD is "gray" (or "grey"), each RGB pixel is converted to an appropriate gray scale value.

If PRINT MODE is "mono" (or "monochromatic"), each non zero value RGB pixel is rendered black.

Note Monochromatic only formatters, map "color" to "gray" and "primary" to "monochromatic". That is, the appropriate monochromatic mode will automatically be chosen.

PIXEL EXPANSION

Indicates the expansion for each pixel on the bitmap and ranges from 1 to 8. For example, to expand each pixel of the bitmap to a 3×3 cell, the expansion is set to 3.

RESOLUTION

Indicates resolutions in dots/inch. This is printer dependent. For example, the HP 3630A has an available graphics resolution of 180 dots/inch, while the LaserJet and LaserJet Plus printers have four resolutions as follows:

- 75 dots/inch
- 100 dots/inch
- 150 dots/inch
- 300 dots/inch

Note that this parameter and the subsequent page_width and page_length parameters are used to determine the output "page" for clipping purposes.

SEND RESOLUTION

If true, specifies that the graphics resolution escape sequence will be sent to the output file (printer). If false, no graphics resolution escape sequence will be sent. This parameter should normally be set to true unless special circumstances exist (such as spooling).

PRINT START POSITION	If this parameter is "current", raster graphics rows start at the current text cursor position. If this parameter is "margin", raster graphics rows start at the left graphics margin. When this parameter is margin, a formfeed is sent to the printer at the completion of the graphics data transfer. If this parameter is current, no formfeed is sent upon completion of the graphics data transfer. Note that the current parameter is only useful for printers (such as the LaserJet or SprintJet) which implement this capability of PCL.
PAGE WIDTH	Specifies the width of the printable graphics area on the page in inches.
PAGE LENGTH	Specifies the length of the printable graphics area on the page in inches.

The symbol # in configuration files starts a comment and is operative for the remainder of the line.

Configuration File Template

A configuration file template for color pcl printers is provided as file:

```
/usr/lib/starbase/formatters/pcl/cfg.ctmplt
```

This template contains values appropriate for the HP 3630A printer. A configuration file template for monochromatic pcl printers is provided as file:

```
/usr/lib/starbase/formatters/pcl/cfg.template
```

A following section details parameter values for supported Hewlett-Packard printers which you may wish to refer to in deciding values for particular fields of your configuration file. Note that parameters are position sensitive. That is, each parameter is required to be present in the form and order listed.

Example Configuration File

```
#-----------------------------------------------------------------
#   ***** Example configuration file for a color PCL printer *****
#-----------------------------------------------------------------
TRUE
   /dev/null
   color
   2
   180
   TRUE
   current
   8.0
   10.5
```

Printer Parameters

This section provides information on each of the printers. It is meant to supplement the documentation provided with your printer. The dots/row column indicates maximum dots per row at maximum density, generally the printer will truncate data exceeding the maximum dots per row. Refer to the applicable printer documentation for the most current information and for dots per row at other than maximum density.

Table PCL-1. Printer Resolution Information

Printer	Resolution	Dots/Row	Comments
HP 2225A	96	640	square pixels only
HP 2227A	96, 192	2536	square pixels only
HP 2228A	96, 192	1536	square pixels only
HP 2235B/D	90, 180	2448	
HP 2686A	75,100,150,300	†	†
HP 256XA/B	70, 140	1848	
HP 293XA/B	90	1024	
HP 3630A	180	1440	
HP C1602A	180	1440/1925‡	

† With the HP 2686A LaserJet and LaserJet Plus printers, the user can specify several different print modes. You should be aware of the following:

‡ With B-sized paper.

1. On the LaserJet, graphics memory is limited to approximately 59 Kbytes. As a result, prints at greater than 75 dots per inch resolution are limited by printer graphics memory. That is, output prints using higher densities are smaller than the paper size. Attempts to print larger images than graphics memory allows will probably cause the printer to display error 20 with unpredictable print results.
2. On the LaserJet Plus graphics memory size is dependent upon previous actions such as down loading of fonts.
3. LaserJet Plus printers may not have enough available graphics memory to handle a full page 300 dots per inch print. See the printer's technical

reference manual for further details. This note does not apply to Laser Jet Plus printers with 2.0 Mbytes of memory.

4. This printer family can dynamically reconfigure graphics memory.
5. The formatter bases print dimensions on page length, page width, and resolution. The results of attempting prints that are not supportable by actual available graphics memory or physical paper size are undefined.

Print Modes

Four print modes are currently supported on color PCL printers. These modes are `color`, `primary`, `gray` (parameter value "`grey`" or "`gray`") and `monochrome` (parameter value "`mono`" or "`monochrome`"). The four modes are explained below:

Print Mode: color

The formatter enables PCL printers to provide the additive (red, green, blue) and subtractive (cyan, yellow, magenta) primary colors. Other colors are generated (by the formatter) by dithering the primary colors. An error diffusion algorithm is utilized to develop the appropriate color cell. Each pixel on the bitmap is expanded into a cell whose size is controlled by the `PIXEL EXPANSION` parameter in the configuration file. Patterns of RGB dots are plotted in the expansion cell to generate a color that the eye perceives as the desired color. The pattern of dots within the expansion cell for each of three planes per row is a fairly complex function of the desired color.

Expansion cell sizes range from 1 to 8. For example, if the size is set to 3, each bitmap pixel is expanded to a 3×3 cell on the plot.

Color mode plotting can take a considerable amount of time depending on the following:

- size of the image.
- number of bitmap planes.
- pixel expansion factor.
- printer interface type.

- error diffusion calculations.

Error Diffusion

The actual intensity of each dot in the output print is determined in a complex manner. The output print is organized into planes (one for each primary additive color). Each plane contains rows of output cells, with each row containing dots equal to the number of source pixels times the `PIXEL EXPANSION` factor (or cell expansion factor).The number of rows in the output is equal to the number of source pixel rows times the `PIXEL EXPANSION` factor. Thus each pixel is expanded to a larger cell in the output according to the `PIXEL EXPANSION` factor.

A color map index value is obtained for the source pixel currently being processed. Residual errors which have accumulated from previously processed output dots are added to the color map index value to obtain a desired color map index value. The desired color map index value is then tested against a value equivalent to half bright. If the desired value is greater than half bright, this output dot will be turned on; otherwise it will be turned off. If this output dot is turned on, a new error value equal to the desired color map index (minus full bright) is accumulated in adjacent output dots. If the output dot is not turned on, only the desired value is accumulated in adjacent dots. The result of this process is that errors in dot intensity are diffused (or accumulated) over adjacent output dots. This process is repeated for each dot being expanded from the source pixel. When the source pixel expansion is complete a new color map index value is obtained for the next source pixel, and the process is repeated.

The error diffusion method works well for most color intensities. Certain color intensities result in generation of unwanted patterns. This is most noticeable with gray (r=g=b) in the range of 0.3 to 0.7. Note that this unwanted pattern problem is discussed in *ACM Transaction on Graphics,* vol. 6, no. 4, October 1987.

Print Mode: color2

The `color2` mode uses the same algorithm as the color mode, with the addition of random noise to each pixel. This random noise breaks up unwanted patterns sometimes seen in large areas of gray. One result of the added random noise is introduction of random (different) color dots, particularly in regions of low luminosity.

Print Mode: primary

While error diffusion is useful for solid images, it is not adequate for line drawings since lines appear intermittent due to "holes" in the dither pattern. The `primary` mode supports direct generation of lines using the primary colors (red, green, blue, cyan, yellow, magenta, black, and white). White lines are mapped to black in `primary` mode. The background is rendered as white.

In `primary` mode, the user's `PIXEL EXPANSION` factor dictates the size of a solid cell for each pixel. `PIXEL EXPANSION` values of 1 to 8 are supported for primary mode. For example, if a bitmap line is green and the configuration file specifies an expansion of 4, then each green bitmap pixel is reproduced by a 4×4 array of green dots.

Print Mode: gray

Gray mode maps each rgb pixel into a gray intensity value according to the YIQ color model. The YIQ color model maps Y into the same chromaticity as luminosity in the CIE color model according to the formula:

```
0.30 * red + 0.59 * green + 0.11 * blue
```

This formatter maps gray intensities into an 8×8 ordered dither pattern providing 65 shades of gray.

Dithering in gray Mode (Half Toning)

The actual intensity of each pixel on the output print is determined in a fairly complex manner. Essentially the output print is organized into 8×8 dither cells (a grid of rows and columns each eight dots across). Then each input pixel is converted from rgb to YIQ yielding an index into a table of ordered dither patterns. Next the input pixel is expanded to a larger cell according to the `PIXEL EXPANSION` parameter. Finally, this cell is copied (tiled) from the ordered dither pattern onto the output page. The actual portion of the 8×8 ordered dither cell pattern copied is determined by the row and column position of the source pixel and output print location. In large areas of similar color the actual dither pattern achieved is 8×8. In areas of rapidly changing color the actual dither pattern achieved may be some smaller size (minimum size = `PIXEL EXPANSION` parameter).

Disappearing Lines in gray Mode

One result of the dithering method used is that single pixel width lines can disappear. When the pixel is copied from the ordered dither pattern (as discussed above) portions of the source pattern are empty (white). With certain conditions the slope of a single pixel line can be such that it intercepts all black or all white pixels in the dither cell locations being copied. This results in a disappearing line. A similar problem results in a line appearing as random size strings of dots.

This mode was designed to be used with solids and polygons rather than with lines. If the bitmap you desire to print consists of lines you should use `monochrome` mode, possibly with no background.

Print Mode: monochrome

The `monochrome` mode maps each nonzero pixel to black. This mode works well for line drawings where a constant (black) intensity is desired for each line. This mode does not work well for solids modeling or filled polygons as every nonwhite pixel maps to black.

Print Mode Differences When Printing Single Planes

The two modes, `gray` and `monochrome` have quite different effects when printing from a bitmap which consists of monochromatic foreground and background. Essentially `gray` mode tries to approximate the actual display as closely as possible in shades of gray. As a result, a display that consists of white text on a black background will be printed faithfully using `gray` mode. That is, the black background will be printed full black while the white letters will not be printed (white being the absence of subtractive color). Conversely, `monochrome` mode will print the foreground (that is, the letters) in black and not print the background. The resulting prints will (correctly) appear to be reversed images of each other.

Using the Graphics Print Procedures

The *Starbase Graphics Techniques* chapter "Storing, Retrieving, and Printing Images" should be read prior to reading this section. Briefly, the graphics print procedures are:

- `bitmap_print`, `dcbitmap_print`—print from a bitmap.
- `file_print`—print from a file created previously using `bitmap_to_file` or `dcbitmap_to_file`.

The user controls the output using parameters of the `bitmap_print` and `file_print` procedures and parameters in the configuration file. Except for the `formatter` and `config` parameters, all other parameters are discussed in the *Starbase Graphics Techniques* manual.

Specifying the Formatter and Config Parameters

The print procedures require specification of two parameters which are unique to the PCL formatter:

1. formatter—The name that is used is "`pcl`".
2. config—This should be set to the desired configuration file.

Using the bmprint Program

The program `bmprint.c` has its source in `/usr/lib/starbase/formatters/pcl`. You may desire to customize it for your application environment or to make multiple copies under different names that reference different configuration files. Essentially this program executes the `gopen` call on a bitmap without initializing it, allowing all or a portion of the bitmap currently displayed to be printed.

Run time parameters allow you to specify:

- start location (`-l` option).
- size of the rectangle (`-s` option).
- rotation of the output print (`-r` option).
- color map mode full or other (`-f` option).
- print the background (`-k` option) (see note below).

- set foreground and background indices (`-c` option).
- set the bitmap bank to print (`-b` option).
- set the display enable mask (`-d`).
- set the plane to print if single plane (`-p`).

You should make a copy of the source, modify it as required to reflect the configuration file and defaults you desire to use, and then compile and link it as described in the section of this chapter concerning linking.

The background (`-k`) option affects the resulting print in one of two ways depending upon whether a single plane is being printed or not. If a single plane is being printed and no background is selected, the foreground and background index parameters are active and specify what is to be printed (foreground) and not printed (background). In all other cases (not single plane), the actual background color index used by the formatter (the index whose printing will be suppressed) is obtained from one of two sources. In the case of a non-single plane being printed from a bitmap opened with the `gopen`, the formatter uses the current Starbase background index. In the case of a non-single plane being printed from a Starbase bitmap file the background index is obtained from the Starbase bitmap file being printed.

A final note on background indexes. If you decide to set the Starbase background color index prior to a `bitmap_print` operation and the color map's shade mode is `CMAP_FULL` with more than eight planes the resulting index is a 24-bit value. The upper eight bits are used for red, the center eight bits are used for blue, and the lower eight bits are used for green. You may want to use `background_color` rather then `background_color_index` providing the specific (float) red, green, and blue values rather than computing the 24-bit index.

```
index25 = (red_index << 16) + (green_index << 8) + blue_index
```

Direct Access Printing

When using direct access the first parameter of the configuration file (output goes to std out parameter) should be set to `FALSE`. The special device file parameter of the configuration file should be set to the special device file of the printer (see the section on setting up the special device file). You must have write permission for the device file.

The following examples assume that a bitmap has been created containing the data you desire to print. In the case of `bitmap_print` and `dcbitmap_print` calls `fildes` is the file descriptor of the bitmap opened with `gopen`. In the case of `file_print`, `myfile.dat` is the Starbase bitmap file previously created. The configuration file is `config.prtr`.

1. Example of `bitmap_print` and `dc_bitmap_print` calls. Refer to the *Starbase Reference* manual for parameter descriptions.

```
bitmap_print(fildes,"pcl","config.prtr",ALL_PLANES,TRUE 0,0.0,1.0,1.0,
        FALSE,1,0,TRUE);

dcbitmap_print(fildes,"pcl","config.prtr",ALL_PLANES,FALSE 0,100,100,
        FALSE,1,0,TRUE);
```

2. Example of a `file_print` call. Refer to the *Starbase Reference* manual for parameter descriptions.

```
file_print(myfile.dat,"pcl","config.prtr",ALL_PLANES,TRUE, 1,0,TRUE);
```

Direct Access Using Redirection or Pipes

Access to a non-spooled printer requires an HP-UX environment (in order to use the ">" and "|" redirection and pipe symbols). The following examples use a configuration file named `config.temp` which has the output to a file named `temp`. The example special device file is `/dev/rp`. The previously prepared Starbase bitmap file is `myfile.dat`.

The HP-UX environment can be obtained from within a program using the *HP-UX Reference,* Section 3 procedure `system`. Similar functionality may be obtained by invoking the *HP-UX Reference,* Section 1 procedure `pcltrans`, or by running the provided `bmprint` program.

1. Example of a `pcltrans` call. Refer to the *Starbase Reference* manual `pcltrans` procedure for parameters.

```
pcltrans myfile.dat <parms> > /dev/rp
```

2. Example of redirection using `file_print`, `cat`, and `system`.

```
file_print(myfile.dat,"pcl","config.temp",ALL_PLANES,TRUE, 1,0,TRUE);
system("cat temp > /dev/rp");
```

3. Example of a `screenpr` call. Refer to the *Starbase Reference* manual `screenpr` procedure for parameters.

```
screenpr -C <parms> > /dev/rp
```

Spooling Examples

Spooling can be done using the *Starbase Reference* procedure `pcltrans`. Spooling may also be done utilizing the `file_print`, `dcbitmap_print`, and `bitmap_print` procedures in conjunction with the *Starbase Reference* procedure `system`. Using the `system` command, you can spool a file from within a program.

A possible sequence within a program might be to create a file using the `bitmap_to_file` procedure and then use the system procedure to invoke the spooler. Alternatively, a program might invoke `bitmap_print` with a configuration file specified that directs output to standard out in a system procedure call which also pipes the output to `lp` in `raw` mode.

The following examples are given as possible ways to spool raster graphics data from Starbase bitmaps. You should review the pertinent sections of the *Starbase Reference* manual for the correct calling parameters (`parms`).

1. Spooling from a Starbase environment (assumes the configuration file sets output to file `myprint.proc` and the Starbase bitmap file name is `myprint.dat`) using `file_print`.

```
/* create the Starbase bitmap file */
  bitmap_to_file( parms..,myprint.dat.. parms);
/* format the file, output filename is myprint.doc */
  file_print(myprint.dat, .. parms);
/* spool the file */
  system("lp -oraw myprint.proc");
```

2. Spooling from a Starbase environment (assumes the configuration file sets output to file `myprint.proc`) using `bitmap_print`.

```
/* format the file - output filename is myprint.proc */
bitmap_print(parms);
system("lp -oraw myprint.proc");
```

3. Spooling using the HP-UX command `screenpr`. The currently displayed bitmap will be spooled.

```
screenpr -C | lp -oraw
```

4. Spooling using the HP-UX command `pcltrans`. Assumes a Starbase bitmap file `myprint.dat` has been previously created.

```
pcltrans myprint.dat | lp -oraw
```

Controlling Print Orientation

The default print orientation is analogous to landscape mode on a LaserJet or LaserJet Plus printer. That is, width is across the long paper dimension, and height is across the narrow paper dimension.

Print Size and Clipping

Print rows that extend beyond the last column on the physical page will generally be clipped by the printer. However, this action is printer dependent.

Print columns that extend beyond the last row on the physical page will be printed onto the next (fanfold) page.

This formatter determines the target page size based on information in the configuration file. Specifically, `page_length`, `page_width`, and `resolution` determine the number of dots in the output page. The `cell_size` parameter is used to determine the number of input pixels that will fit on the output page as follows:

```
output pixels across = page_width * resolution / cell_size
output pixels down = page_length * resolution / cell_size
```

Prints will be truncated according to the target page size by the formatter. The following example may help clarify this.

```
Request to print the entire frame buffer

Source frame buffer width = 1280 pixels
Source frame buffer height = 1024 pixels

Page width = 8.0 in
Page length = 10.5 in
Resolution = 300 dots per inch
```

```
cell_size = 2
available output pixels across = 8.0 * 180 / 2 = 720
available output pixels down = 10.5 * 180 / 2 = 945
result -- 720 < 1280 and 945 < 1024 -- truncate in both dimensions

    cell size = 1 then
    available output pixels across = 8.0 * 180 / 1 = 1400
    available output pixels down = 10.5 * 180 / 1 = 1890
    result -- 1400 > 1280 and 1890 > 1024 -- no truncation
```

Linking and Running Your Program

The PCL formatter `pcl_fmt` is located in `/usr/lib/starbase/formatters/pcl` with the file name `libfmtpcl.a`. The PCL formatter requires `fmt_table.o` (which associates the formatter and name) to be present at run time. Hewlett-Packard provides the source as `/usr/lib/starbase/formatters/fmt_table.c`. The Starbase procedures `bitmap_print` and `file_print` require a Starbase environment and a device opened with the Starbase `gopen` call. The following example (`myprog.c` being your program name) uses the `-l` option for the Starbase and I/O libraries. This example also uses the `-l` option for a representative driver `-ldd3001`. To compile and link a program using `bitmap_print` or `file_print` along with other Starbase procedures requiring a device opened with the Starbase call `gopen`, use:

```
cc myprog.c /usr/lib/starbase/formatters/fmt_table.c  \
/usr/lib/starbase/formatters/pcl/libfmtpcl.a  \
-ldd3001 -lsb1 -lsb2 -ldvio -lm -o myprog

fc myprog.f /usr/lib/starbase/formatters/fmt_table.c \
/usr/lib/starbase/formatters/pcl/libfmtpcl.a \
-ldd3001 -lsb1 -lsb2 -ldvio -lm -o myprog

pc myprog.p /usr/lib/starbase/formatters/fmt_table.c \
/usr/lib/starbase/formatters/pcl/libfmtpcl.a \
-ldd3001 -lsb1 -lsb2 -ldvio -lm -o myprog
```

To compile and link `/usr/lib/starbase/formatters/pcl/bmprint.c`, the following sequence can be used. As explained previously, `screenpr` allows the user to print a currently displayed bitmap.

```
cc /usr/lib/starbase/formatters/pcl/bmprint.c \
/usr/lib/starbase/formatters/fmt_table.c \
/usr/lib/starbase/formatters/pcl/libfmtpcl.a \
-ldd300l -lsb1 -lsb2 -ldvio -o bmprint
```

Warning and Error Messages

This section discusses warning and error messages provided by the PCL formatter.

Warning Messages

`Unrecognized item in config file, line` *xx*

This indicates that a problem existed in the configuration file with the parameter at line *xx*. This warning will be followed with an error message indicating an error reading configuration file.

`Print truncated`

This indicates the formatter determined the print too large to fit in the print space defined by resolution, page length, and page width. The formatter then truncated the print to fit the print space defined.

Error Messages

`Raster formatter specified is not in table`

This indicates that the formatter specified was not found in `fmt_table.o`.

You should check that `/usr/lib/starbase/formatters/fmt_table.c` contains the `pcl` entry, and that `/usr/lib/starbase/formatters/fmt_table.o` was included in your link sequence.

`Device is not bitmap`

This indicates the bitmap specified in a `bitmap_print` or `dcbitmap_print` call was not a bitmap opened with `gopen`.

`Plane number is out of range`

This indicates that the single plane specified for printing was not in the specified bitmap. This error can occur with `bitmap_print`, `dcbitmap_print` or `file_print` calls.

`Cannot open source file`

This indicates that the Starbase bitmap file specified in a `file_print` call could not be opened.

`Specified source file not bitmap data`

This indicates that the file specified in a `file_print` call was successfully opened; however, it was not a Starbase bitmap file.

`Error in closing raster file`

This indicates a problem in closing the Starbase bitmap file.

`Unable to open configuration file`

This indicates that the configuration file specified in a `file_print`, `bitmap_print` or `dc_bitmap` print could not be opened.

`Error while reading configuration file`

This indicates a parameter problem in a successfully opened configuration file.

`Error while opening output file:` *xxxx*

This indicates a problem opening the special device file or output file that was specified in the configuration file with output not to standard out.

`Unable to allocate input buffer`

This indicates a malloc call (to allocate 64K bytes) failed. You will need to provide more memory for the formatter. In the case of a source bitmap which contains multiple banks a total of 196K bytes of input buffer space will be required.

`Unable to allocate output buffer`

This indicates a malloc call, (to allocate 92K bytes) failed. You will need to provide more memory for the formatter.

`Unable to allocate color table buffer`

This indicates a malloc call (to allocate 512 bytes) failed. You will need to provide more memory for the formatter.

`Formatter internal error`. All locations except 31

This indicates a problem internal to the formatter code. The most likely cause is failure of a malloc call (to allocate 64K bytes for processing source data).

`Formatter internal error`. Location 31

This indicates a single plane bitmap file was used with a full depth formatter call. In the case of `file_print`, `print_mode` was negative or `ALL_PLANES` instead of the plane number contained in the file. In the case of `pcltrans`, the `-pplane` option was not used or was used incorrectly.

Setting Up the Spooler

The steps for setting up the graphics spooler are very similar to the steps for configuring the LP spooler system. Refer to the *HP-UX System Administrator Manual* (the "System Administrator's Toolbox" section) for details.

1. The LP spooler system needs to use HP-UX 1.1 (for Series 800) or 5.2 or later (for Series 300) printer models.
2. Always use the raw mode (`-oraw`) of the lp spooler.
3. You must have write access.
4. First make sure that the lp spooler works with text. If you have problems refer to the *HP-UX System Administrator Manual.*
5. If in a Starbase program environment, you should make the *HP-UX Reference,* section 3 procedure call `system` with the appropriate string containing the necessary files, parameters, etc.
6. If you desire to print a currently displayed bitmap you may use a version of `bmprint` redirected or piped as required.

Special Considerations for Non-Spooled Serial Output

The normal `stty` settings for an unopened serial device may not correspond with the desired `stty` settings. For example, the default baud rate is 300. The following information for setting up the special device file and then setting `stty` is provided as a starting point for your own requirements.

1. Typical `mknod` for an HP 3630A at select code 9

```
# mknod /dev/rp c 1 0x090004
```

2. To configure the port for normal printing

```
stty -parenb -ineqak cs8 9600 -cstopb \
    -clocal ixon opost onlcr tab3 < /dev/rp
```

3. To configure the port for raster printing, execute the following `stty` commands (or equivalent `ioctl(2)` calls).

```
stty -onlcr -opost -tabdly < /dev/rp
```

PCL-IMAGING

Printer Command Language Imaging Formatter

Overview

The PCL Imaging Formatter is a superset of the PCL Formatter. As such, the PCL Formatter (see "PCL" chapter) will drive a device that supports the imaging extentions of PCL (with reduced performance). However, the PCL Imaging Formatter will not drive a device that supports straight PCL with no imaging capabilities.

Read the chapter on "Storing, Retrieving, and Printer Images" in the *Starbase Graphics Techniques* manual before reading this chapter.

The PCL Imaging Formatter permits hard copies from bitmaps or Starbase bitmap file to a color or monocromatic printer that supports the imaging extension of PCL. If a device supports the imaging extensions of PCL, it is able to process the raw bitmap and raw color map data internally, creating the fully processed image without the help of the host computer. Devices that do not support these capabilities rely on the host computer to perform all the image processing, treating the printer as a dumb PCL device. These imaging extensions give an increase in performance, image quality, and image processing options.

You can create hard copies using this formatter in the following ways.

- The HP-UX command `pcltrans` (see the *Starbase Reference* manual for options) is used to print a previously created Starbase bitmap file. Starbase bitmap files are created using the Starbase function `bitmap_to_file` or `dcbitmap_to_file`.
- The HP-UX command `screenpr` (see the *Starbase Reference* manual for options) is used to print a currently displayed bitmap. This command reads the display's image planes and current hardware color map.

  The `screenpr` command is supported on displays that use the following Starbase device drivers:

Series 300: `hp3001, hp300h, hp98550, hp98556, hp98720, hp98721, hp98730, hp98731.`

Series 800: `hp98550, hp98556, hp98720, hp98721, hp98730, hp98731.`

Both the `pcltrans` and `screenpr` commands are capable of using or not using the PCL Imaging Formatter. If you wish to use the PCL Imaging Formatter, as opposed to the PCL Formatter (see PCL Formatter chapter), you must pass in a special option that states the device you are using supports the imaging extensions of PCL.

Currently, the PCL Imaging Formatter is not available through the Starbase functions `bitmap_print` and `file_print`. If you invoke either of these functions while running a Starbase program, the image will be processed entirely by the host computer, not by the device's image processing software.

Key Points of the PCL Imaging Formatter

1. The PCL Imaging Formatter supports HP C1602A (PaintJet XL)
2. You can print in gray scale, monochromatic (black and white), primary (red, green, blue, cyan, yellow, magenta, black and white), or in color using the following color algrithms:

   error diffusion

   ordered dither
3. Prints can be sized using non-integer pixel scaling. The default size is the entire size of the paper used in the printer. Print size can also be determined by specifying destination dimensions in inches.
4. Prints can be gamma-corrected by selecting one of the printer's built-in gamma correction curves.
5. The PCL Imaging Formatter is not a Starbase driver—you do not do moves, draws, etc. to the printer. Instead, you process currently displayed bitmaps or previously created Starbase bitmap files for output to the printer.
6. The PCL Imaging Formatter works with HP-UX releases 7.0 and later for both the Series 300 and Series 800 computers.

Printer Configurations

There are two fundamental printer configurations of interest, spooled and non-spooled. The primary difference between the two configurations is that spooling uses the system spooler (`lp`) in the raw (`-oraw` option) mode. This section gives an overview of these configurations so you can choose the appropriate method for your application.

Non-Spooled Operation

The only non-spooled operation currently supported by the PCL Imaging Formatter is direct access printing in the HP-UX environment. Direct access printing involves a non-shared printer directly connected to the host system. The standard output (`stdout`) from the HP-UX commands `pcltrans` and `screenpr` is "piped" to the printer's special device file (see the section on "Setting Up the Special Device File" in this chapter). You will need to have write permission for the special file.

The HP-UX environment can be obtained from within a program using the procedure `system` (described in *HP-UX Reference,* Section 3).

Direct Access Printing

Examples:

- Using a bitmap file from a Starbase program:

```
/* Create the starbase bitmap file */

bitmap_to_file(params ... ,myprint.bit, ... params);

/* Process the file in color and send to the printer */

system("pcltrans -I -C myprint.bit > /dev/rp");
```

- Print currently displayed bitmap from a program in color:

```
system("screenpr -I -C -F/dev/crt > /dev/rp");
```

- Print currently displayed window from a program in color:

```
*/ Origin=10,10, Width=100, Height=200 */

system("screenpr -I -C -X10 -Y10 -D100 -H200 -F/dev/crt > /dev/rp">;
```

- Print color bitmap file using ordered dither in an HP-UX environment:

```
pcltrans -I -a3 myprint.bit > /dev/rp
```

- Print currently displayed bitmap in gray scale, rotated:

```
screenpr -I -a5 -R -F/dev/crt > /dev/rp
```

Spooled Operation

Spooled operation is the best mode if you have a shared printer. The HP-UX commands `pcltrans` and `screenpr` can also be utilized in a spooled environment (see the *Starbase Reference* manual for details on `pcltrans` and `screenpr`).

`pcltrans` is used as a filter to process a Starbase bitmap file previously created by the `bitmap_to_file` procedure. The `stdout` (standard out) is then piped to the `lp` spooler in raw mode. Spooling can be done locally, or the `pcltrans` command output can be piped into a file and sent to a remote printer on another computer.

`screenpr` is used to process a currently displayed bitmap. Its output is also sent to `stdout` and can be piped to either the `lp` spooler or a file for remote printing.

Spooled Printing

Examples:

- Using a bitmap file from a starbase program:

```
/* Create the starbase bitmap file */

bitmap_to_file(params ... ,myprint.bit, ... params);

/* Process the file in color and send to the printer */

system("pcltrans -I -C myprint.bit | lp -oraw");
```

- Print currently displayed bitmap from a program in color:

```
system("screenpr -I -C -F/dev/crt | lp -oraw");
```

- Print currently displayed window from a program in color:

```
/* Origin=10,10, Width=100, Height=200 */
```

```
system("screenpr -I -C -X10 -Y10 -D100 -H200 -F/dev/crt | lp -oraw");
```

- Print color bitmap file using ordered dither in an HP-UX environment:

```
pcltrans -I -a3 myprint.bit > myprint.out
lp -oraw myprint.out
```

or

```
cat myprint.bit | pcltrans -I -a3 | lp -oraw
```

- Print currently displayed bitmap in gray scale, rotated:

```
screenpr -I -a5 -R -F/dev/crt | lp -oraw
```

Spooler Conflicts

In the following scenarios, interleaved/unusable output may be produced:

- Direct access output mode is used for a printer which is currently used for spooling via the `lp` command.
- More than one person is using direct access printing mode on the same device.

In general, if a device is configured for spooling with the `lp` command, all graphics output should be done using the spooling print mode. Only use non-spooled (direct access) print mode when a device is not shared. Simultaneous usage of spooled and non-spooled modes should be avoided.

Software Structure

The following files are used for color/monochromatic printing on printers that support the imaging extensions of PCL:

`/user/bin/pcltrans`

`/user/bin/screenpr`

Setting Up the Special Device File

To directly access a printer, you need a special device file. If the printer has already been assigned to a node as a system printer, you may use that device file (you must have write permission on that device file).

If a special device file for your printer has *not* been assigned, the `mknod` command must be performed before proceeding. For this you must be super-user. Enter the select code in hexadecimal format (for example, a select code of 22 = 16 Hex).

Series 300 Computers

HP-IB printer: select code 7 (internal HP-IB), bus address 3 (determined by device), raw bit set:

```
mknod /dev/rp c 21 0x070301
```

Serial printer: select code 9, port address 0 (determined by type of interface card in system), direct connect bit set:

```
mknod /dev/rp c 1 0x090004
```

Series 800 Computers

HP-IB printer: ⟨*Lu*⟩ is the hardware logical unit, ⟨*Ad*⟩ is the port address:

```
mknod /dev/rp c 21 0x00⟨Lu⟩⟨Ad⟩
```

Serial printer: ⟨*Lu*⟩ is the hardware logical unit, ⟨*Ad*⟩ is the port address:

```
mknod /dev/rp c 1 0x00⟨Lu⟩⟨Ad⟩
```

You may need to set owner, group, and mode (`chown`, `chgrp`, and `chmod`) appropriately.

Refer to the *HP-UX System Administrator Manual* for details on using the `mknod` command.

Configuration Files

The PCL Imaging Formatter currently is unsupported through the Starbase functions `bitmap_print` and `file_print`; therefore, no configuration files are necessary (see the "PCL Formatter" chapter for a description of configuration files).

Printer Parameters

The only printer supported by the PCL Imaging Formatter is the PaintJet XL (C1602A). The printer is 180 dots per inch and can produce output on both A-size paper and B-size paper.

Print Modes

There are seven print algorithms on printers that implement the imaging extensions of PCL:

Table PCL-IMAGING–1. Print Modes

Selection	Algorithm
0	no algorithm
1	snap to primaries
2	snap to black & white
3	color ordered dither
4	color error diffusion
5	monochrome ordered dither
6	monochrome error diffusion

They are selected in the `pcltrans` and `screenpr` commands by the `-a` option. Each mode is explained in the following sections.

Snap to Primaries

While error diffusion is useful for solid images, it is not adequate for line drawings since lines appear intermittent due to "holes" in the dither pattern. The `primary` mode supports direct generation of lines using the primary colors (red, green, blue, cyan, yellow, magenta, black, and white). White lines are mapped to black in `primary` mode. The background is rendered as white.

Snap to Black and White

The `monochrome` mode maps each nonzero pixel to black. This mode works well for line drawings where a constant (black) intensity is desired for each line. This mode does not work well for solids modeling or filled polygons as every nonwhite pixel maps to black.

Color Ordered Dither or Monochrome Ordered Dither

In order dither, the intensity of each point (x,y) in a pixel matrix depends on the desired intensity at that point I(x,y) and an 8×8 dither matrix. The value of each cell (i,j) in the dither matrix is computed by:

```
i = x modulo 8
j = y modulo 8
```

If I(x,y) > D(i,j), the point corresponding to the (x,y) is intensified; otherwise, it is not.

Color Error Diffusion or Monochrome Error Diffusion

The actual intensity of each dot in the output print is determined in a complex manner. A color map index value is obtained for the source pixel currently being processed. Residual errors which have accumulated from previously processed output dots are added to the color map index value to obtain a desired color map index value. The desired color map index value is then tested against a value equivalent to half bright. If the desired value is greater than half bright, this output dot will be turned on; otherwise it will be turned off. If this output dot is turned on, a new error value equal to the desired color map index (minus full bright) is accumulated in adjacent output dots. If the output dot is not turned on, only the desired value is accumulated in adjacent dots. The result of this process is that errors in dot intensity are diffused (or accumulated) over adjacent output dots. This process is repeated for each dot being expanded from

the source pixel. When the source pixel expansion is complete a new color map index value is obtained for the next source pixel, and the process is repeated.

The error diffusion method works well for most color intensities. Certain color intensities result in generation of unwanted patterns. This is most noticeable with gray (r=g=b) in the range of 0.3 to 0.7. Note that this unwanted pattern problem is discussed in *ACM Transaction on Graphics,* vol. 6, no. 4, October 1987.

Disappearing Lines in Monochromatic Ordered Dither

One result of the dithering method used is that single pixel width lines can disappear. When the pixel is copied from the ordered dither pattern (as discussed above) portions of the source pattern are empty (white). With certain conditions the slope of a single pixel line can be such that it intercepts all black or all white pixels in the dither cell locations being copied. This results in a disappearing line. A similar problem results in a line appearing as random size strings of dots.

This mode was designed to be used with solids and polygons rather than with lines. If the bitmap you desire to print consists of lines you should use `monochrome` mode, possibly with no background.

Differences When Printing Single Planes

The two modes, snap to black and white and monochrome ordered dither have quite different effects when printing from a bitmap which consists of monochromatic foreground and background. Essentially monochromatic ordered dither mode tries to approximate the actual display as closely as possible in shades of gray. As a result, a display that consists of white text on a black background will be printed faithfully. That is, the black background will be printed full black while the white letters will not be printed (white being the absence of subtractive color). Conversely, snap to black and white mode will print the foreground (that is, the letters) in black and not print the background. The resulting prints will (correctly) appear to be reversed images of each other.

Controlling Print Orientation

The default print orientation is left to right across the length of the paper (equivalent to the LaserJet landscape mode). You can cause the HP-UX commands `pcltrans` and `screenpr` orient the print across the width of the paper by using the `-R` option.

Print Sizing and Clipping

By default, the printer will scale the image to the size of the paper used. However, the user can select the destination size of the image by using the `-x`, `-y`, `-d`, `-h` options of the `pcltrans` and `screenpr` HP-UX commands.

Paper size is automatically sensed by the printer. You can manually define the paper size by using the `pcltrans` and `screenpr` options `-l`, `-w`.

Refer to the *Starbase Reference* manual for detailed option information on the `pcltrans` and `screenpr` HP-UX commands.

Warnings and Error Messages

Refer to the "PCL Formatter" chapter for details on warnings and error messages.

Contents

GESC

Gescapes

Introduction

This appendix provides information concerning the ⟨*op*⟩, `arg1` and `arg2` parameters used with the gescape functions common to two or more device drivers. Those `gescape` functions unique to a specific device driver are discussed in the appropriate driver section of this manual.

The `gescape` function allows the application program to input or output to a device in a device dependent manner. The term `gescape` is derived from "graphics escape" and is analogous similar escape functions supported by other graphics libraries. The syntax for the `gescape` function is:

```
gescape (fildes, op, arg1, arg2)
```

`fildes` is the file descriptor of the device to be accessed (returned by the Starbase call `gopen`).

⟨*op*⟩ is the "operation code" (opcode) which specifies the device dependent action to be performed.

`arg1` and `arg2` are two parameters (pointers to argument lists) which provide the information needed by `gescape` to do the desired job.

Table GESC-1. Supported Operation Codes (op)

⟨*op*⟩ Parameter	Function
BLINK_INDEX	Alternate Between HP 98720 or HP 98730 Hardware Color Maps.
BLINK_PLANES	Blink the display using a mask.
GR2D_DEF_MASK	Defines mask for 3-operand raster operation.
GR2D_FILL_PATTERN	Define 16×16 dither and fill pattern.
GR2D_MASK_ENABLE	Enables 3-operand raster operation.
GR2D_MASK_RULE	Set 3-operand drawing mode.
GR2D_OVERLAY_TRANSPARENT	Turns on/off transparency of 0 pixels.
GR2D_PLANE_MASK	Overrides the mask.
GR2D_REPLICATE	Allows square pixel replication.
HPGL_READ_BUFFER	Allows you to read data from the device
HPGL_SET_PEN_NUM	Set plotter number of pens.
HPGL_SET_PEN_SPEED	Set plotter pen velocity.
HPGL_SET_PEN_WIDTH	Set plotter pen width.
HPGL_WRITE_BUFFER	Permits direct communication of HP-GL commands to supported devices.
IGNORE_RELEASE	Trigger when button pressed.
IMAGE_BLEND	Enable/disable video blending.
LS_OVERFLOW_CONTROL	Sets options for light source overflow situations.
OVERLAY_BLEND	Control blending of overlay plane frame buffer.

Table GESC-1. Supported Operation Codes (op) Continued

⟨*op*⟩ Parameter	Function
PATTERN_FILL	Fills polygon with stored pattern.
PAN_AND_ZOOM	Pixel and zoom.
R_BIT_MASK	Identifies the plane(s) to read to or write from.
R_BIT_MODE	Changes the `raw` mode flag.
R_DEF_ECHO_TRANS	Define raster echo transparency.
R_DEF_FILL_PAT	Defines the current 4×4 pixel dither cell.
R_DMA_MODE	Changes the definition of the `raw` flag for block writes.
R_ECHO_CONTROL	Control hardware cursor allocation.
R_ECHO_FG_BG_COLORS	Define color attributes.
R_ECHO_MASK	Define cursor mask.
R_FULL_FRAME_BUFFER	Allows access to the off screen area of the frame buffer.
R_GET_FRAME_BUFFER	Reads the frame buffer and control space addresses.
R_GET_WINDOW_INFO	Returns frame buffer address of Windows/9000 window.
R_LINE_TYPE	Define line style and repeat length.
R_LOCK_DEVICE	Locks the specified device.
R_OFFSCREEN_ALLOC	Allocates offscreen frame buffer memory.
R_OFFSCREEN_FREE	Frees allocated offscreen frame buffer memory.
R_OV_ECHO_COLORS	Select overlay echo colors.
R_OVERLAY_ECHO	Select plane to contain cursor.

Table GESC–1. Supported Operation Codes (op) Continued

⟨*op*⟩ Parameter	Function
R_TRANSPARENCY_INDEX	Specify transparency index.
R_UNLOCK_DEVICE	Unlocks the specified device.
READ_COLOR_MAP	Reads the color map.
SET_BANK_CMAP	Install frame buffer bank color maps.
SWITCH_SEMAPHORE	Controls the device access semaphores.
TRANSPARENCY	Allows "screen door" for transparency pattern.
TRIGGER_ON_RELEASE	Trigger when button released.
ZWRITE_ENABLE	Allows creation of 3D cursors in overlay.

Table GESC-2. Supported Device Drivers

⟨*op*⟩ Parameter	Supported Drivers
BLINK_INDEX	hp98720, hp98721, hp98730, hp98731
BLINK_PLANES	(blink speed 2.4 Hz) hp3001, hp300h, hp98700, hp98710 (blink speed 3.75 Hz) hp98720, hp98720w, hp98721,hp98550, hp98556, hp98730, hp98731
GR2D_DEF_MASK	hp98550, hp98556
GR2D_FILL_PATTERN	hp98550, hp98556
GR2D_MASK_ENABLE	hp98550, hp98556
GR2D_MASK_RULE	hp98550, hp98556
GR2D_OVERLAY_TRANSPARENT	hp98550, hp98556
GR2D_PLANE_MASK	hp98550, hp98556
GR2D_REPLICATE	hp98550, hp98556
HPGL_READ_BUFFER	CADplt, CADplt2
HPGL_SET_PEN_NUM	hpgl, CADplt
HPGL_SET_PEN_SPEED	hpgl, CADplt
HPGL_SET_PEN_WIDTH	hpgl, CADplt
HPGL_WRITE_BUFFER	hpgl, CADplt, CADplt2
IGNORE_RELEASE	hphil, Windows/9000
IMAGE_BLEND	hp98730, hp98731
LS_OVERFLOW_CONTROL	hp98721, hp98731
OVERLAY_BLEND	hp98730, hp98731
PAN_AND_ZOOM	hp98730, hp98731
PATTERN_FILL	hp98721, hp98731
R_BIT_MASK	hp3001, hp300h, hp98700, hp98710, hp98720, hp98720w,hp98721, hp98550, hp98556, h98730, hp98731

Table GESC–2. Supported Device Drivers Continued

⟨*op*⟩ Parameter	Supported Drivers
R_BIT_MODE	hp300l, hp300h, hp98700, hp98710, hp98720, hp98720w,hp98721, hp98550, hp98556, hp98730, hp98731
R_DEF_ECHO_TRANS	hp98720, hp98721, hp98550, hp98730, hp98731
R_DEF_FILL_PAT	hp300l, hp300h, hp98700, hp98710, hp98720, hp98720w,hp98721, hp98550, hp98730, hp98731
R_DMA_MODE	hp98730, hp98731 Models 825 and 835 SPUs with an A10474 interface card
R_ECHO_CONTROL	hp98730, hp98731
R_ECHO_FG_BG_COLORS	hp98730, hp98731
R_ECHO_MASK	hp98730, hp98731
R_FULL_FRAME_BUFFER	hp300l, hp300h, hp9836a, hp98700, hp98710, hp98720,hp98720w, hp98721, hp87550, hp98556, hp98730, hp98731
R_GET_FRAME_BUFFER	hp300l, hp300h, hp9836a, hp98700, hp98710, hp98720,hp98720w, hp98721, hp98550, hp98556, hp98730, hp98731
R_GET_WINDOW_INFO	hp300h, hp300l, hp98700, hp98720w, window/9000 hp98730, hp98731
R_LINE_TYPE	hp98720, hp98721, hp98730, hp98731, SMD
R_LOCK_DEVICE	hp300l, hp300h, hp98700, hp98720, hp98720w, hp98721, hp98550, hp98556, hp98730, hp98731, window/9000

Table GESC-2. Supported Device Drivers
Continued

⟨*op*⟩ Parameter	Supported Drivers
R_OFFSCREEN_ALLOC	hp98550, hp98556, hp98730, hp98731
R_OFFSCREEN_FREE	hp98550, hp98556, hp98730, hp98731
R_OV_ECHO_COLORS	hp98720, hp98721, hp98730, hp98731
R_OVERLAY_ECHO	hp98720, hp98721, hp98550, hp98556, hp98730
R_TRANSPARENCY_INDEX	hp98720, hp98721, hp98730, hp98731
R_UNLOCK_DEVICE	hp300l, hp300h, hp98700, hp98720, hp98720w, hp98721, hp98550, hp98556, hp98730, hp98731, Windows/9000
READ_COLOR_MAP	hpterm, hp300l, hp300h, hp98700, hp98710, hp98720, hp98720w,hp98721, hp98550, 987556, hp98730, hp98731
SET_BANK_CMAP	hp98730, hp98731
SWITCH_SEMAPHORE	hp300l, hp300h, hp9836a, hp98700, hp98710, hp98720,hp98720w, hp98721, hp98550, hp98556, hp98730, hp98731,Windows/9000
TRIGGER_ON_RELEASE	hp-hil, Windows/9000
TRANSPARENCY	hp98721, hp98731
ZWRITE_ENABLE	hp98721, hp98731

BLINK_INDEX

The ⟨*op*⟩ parameter is BLINK_INDEX.

The HP 98720, HP 98721, HP 98730, and HP 98731 have two separate hardware color maps. They alternate between the two color maps: one color map is used for 133 ms (milliseconds) and the other color map is used for 133 ms. When the color table is changed either by INIT or by **define_color_map** both hardware color maps are updated to the same software color table. This **gescape** allows the user to set a color map value in only one hardware color map. The color set by the **gescape** goes directly into one of the hardware color maps and does not effect the Starbase software color table. The effect will be that a single color map index will blink between the color set in the Starbase color table and the color set by this **gescape**. The color map value set with this **gescape** will be overwritten any time Starbase updates that entry in its software color table.

The **arg1** parameter contains the index number, red value, green value, and blue value in that order.

The **arg2** parameter is ignored.

The example given below will blink index 5 between the color value given in the Starbase color table and red.

When in CMAP_FULL mode, the index number can contain three index values simultaneously. The index value for red is in byte 2, the index number for green is in byte 1, and the index value for blue is in byte 0.

When video blending is enabled on the HP 98730 or HP 98731, color map index blinking will not be operative. See the description of the **gescape** IMAGE_BLEND for more detail.

To blink color map planes see the **gescape** for BLINK_PLANES.

C Syntax

```
/* gescape_arg is typedef defined in starbase.c.h */

gescape_arg arg1, arg2;
   :
arg1.f[0]=5.0;
arg1.f[1]=1.0;
arg1.f[2]=0.0;
```

```
arg1.f[3]=0.0;
gescape(fildes,BLINK_INDEX,&arg1,&arg2);
```

FORTRAN77 Syntax

```
real arg1(64),arg2(64)
arg1(1)=5.0
arg1(2)=1.0
arg1(3)=0.0
arg1(4)=0.0
call gescape(fildes,BLINK_INDEX,arg1,arg2)
```

Pascal Syntax

```
{gescape_arg is defined in starbase.p1.h}

var
   arg1,arg2:gescape_arg;
      :
begin
   arg1.f[1] := 5.0;
   arg1.f[2] := 1.0;
   arg1.f[3] := 0.0;
   arg1.f[4] := 0.0;
   gescape(fildes,BLINK_INDEX,arg1,arg2);
```

BLINK_PLANES

The ⟨*op*⟩ parameter is BLINK_PLANES.

This gescape allows you to blink the display. To blink individual color map indexes refer to the R_BLINK_INDEX segment of those device driver sections that allow color map blinking.

The following drivers support a blink speed of 2.4 Hertz:

hp300l hp300h hp98700 hp98710

The following drivers support a blink speed of 3.75 Hertz:

hp98720 hp98720w hp98721 hp98550 hp98556 hp98730 hp98731

The arg1 parameter is a mask indicating which planes to blink. The arg1 parameter can be any value from 0–255. For example, if arg1 is 5, planes 0 and 2 of the device will blink.

Devices which support video blending allow individual blink control for all planes when blending is enabled. In this case arg1 can contain values with more than eight bits. See the description of the gescape IMAGE_BLEND for more details.

The arg2 parameter is ignored.

Exceptions - HP 98720w

This gescape works only in image plane configuration; that is, when this driver is running to the image planes and not the overlay planes.

C Syntax

```
/* gescape_arg is typedef defined in starbase.c.h */

gescape_arg arg1, arg2;
      :
   arg1.i[0]=5;
   gescape(fildes,BLINK_PLANES,&arg1,&arg2);
```

FORTRAN77 Syntax

```
integer*4 arg1(64),arg2(64)
arg1(1)=5
call gescape(fildes,BLINK_PLANES,arg1,arg2)
```

Pascal Syntax

```
{gescape_arg is defined in starbase.p1.h}

var
   arg1,arg2:gescape_arg;
      :
begin
   arg1.i[1] := 5;
   gescape(fildes,BLINK_PLANES,arg1,arg2);
```

GR2D_DEF_MASK

The ⟨*op*⟩ parameter is GR2D_DEF_MASK.

The HP 98548A, HP 98549A, HP 98550A, and HP 319C displays, and the HP 98556A accelerator, have hardware capability for three-operand raster combinations: that is, operations that use a tiling mask and a replacement rule that specify the combination of the source, mask, and destination. When enabled, the mask rule and current mask are used for **block_write** and **block_move** operations. When disabled, the normal replacement rule (see **drawing_mode**) is used and the current mask is ignored.

The hardware only allows a mask size of 16×16 pixels. This mask repeats over the entire screen area. The mask is full-depth (that is, it is specified as a byte per pixel, and as many low-order bits are significant as there are color planes in the display being accessed). Each plane of the mask is applied only to the corresponding source and destination planes.

Related **gescape** functions are GR2D_MASK_RULE and GR2D_MASK_ENABLE.

This **gescape** allows you to define the mask to be used. The **arg1** parameter contains 256 bytes in row-major order, representing the mask (16 pixels wide by 16 pixels high).

The **arg2** parameter is ignored.

This mask remains in effect for three-operand combinations until this **gescape** is used again to set another mask. The default mask is all ones.

The following example sets the mask to a checkerboard of 8 8 squares in the first plane.

C Syntax

```
/* gescape_arg is typedef defined in starbase.c.h */

gescape_arg arg1, arg2;
int row;
   :
for (row=0; row<8; row++)
{
   arg1.i[row*4] = 0x01010101;
   arg1.i[row*4+1] = 0x01010101;
   arg1.i[row*4+2] = 0;
   arg1.i[row*4+3] = 0;
}
for (row=8; row<16; row++)
{
   arg1.i[row*4] = 0;
   arg1.i[row*4+1] = 0;
   arg1.i[row*4+2] = 0x01010101;
   arg1.i[row*4+3] = 0x01010101;
}

gescape(fildes,GR2D_DEF_MASK,&arg1,&arg2);
```

FORTRAN77 Syntax

```
      integer*4 arg1(64),arg2(1), row

      do 100 row=0,7
         arg1(row*4+1) = Z'01010101';
         arg1(row*4+2) = Z'01010101';
         arg1(row*4+3) = 0;
         arg1(row*4+4) = 0;
100   continue
      do 200 row=8,15
         arg1(row*4+1) = 0;
         arg1(row*4+2) = 0;
         arg1(row*4+3) = Z'01010101';
         arg1(row*4+4) = Z'01010101';
200   continue

      call gescape(fildes,GR2D_DEF_MASK,arg1,arg2)
```

Pascal Syntax

```
{gescape_arg is defined in starbase.p1.h}

var
   arg1,arg2:gescape_arg;
   row: integer;
      :
begin
   for row := 0 to 7 do begin
      arg1.i[row*4+1] := hex('01010101');
      arg1.i[row*4+2] := hex('01010101');
      arg1.i[row*4+3] := 0;
      arg1.i[row*4+4] := 0;
   end;
   for row := 8 to 15 do begin
      arg1.i[row*4+1] := 0;
      arg1.i[row*4+2] := 0;
      arg1.i[row*4+3] := hex('01010101');
      arg1.i[row*4+4] := hex('01010101');
   end;

   gescape(fildes,GR2D_DEF_MASK,arg1,arg2);
```

GR2D_FILL_PATTERN

The ⟨*op*⟩ parameter is GR2D_FILL_PATTERN.

This gescape allows the user to define a 16×16 dither or fill pattern that will be used as the source fill for polygons and rectangles. The bytes defining the pattern are passed to the driver through arg1. The 256 bytes are placed in the fill pattern cell in row major order. After gescape is called the polygon and rectangle primitives will be filled with the user-defined pattern until another pattern is defined with gescape or until the fill is redefined using interior_style and pattern_define.

This gescape is provided for compatibility with older device drivers. It is suggested that the INT_PATTERN interior style be used instead of this gescape.

The arg2 parameter is ignored.

The following example defines a checkerboard fill pattern.

C Syntax

```
/* gescape_arg is typedef defined in starbase.c.h */

gescape_arg arg1, arg2;
int row;
    :
for (row=0; row<8; row++)
{
   arg1.i[row*4] = 0x01010101;
   arg1.i[row*4+1] = 0x01010101;
   arg1.i[row*4+2] = 0;
   arg1.i[row*4+3] = 0;
}
for (row=8; row<16; row++)
{
   arg1.i[row*4] = 0;
   arg1.i[row*4+1] = 0;
   arg1.i[row*4+2] = 0x01010101;
   arg1.i[row*4+3] = 0x01010101;
}

gescape(fildes,GR2D_FILL_PATTERN,&arg1,&arg2);
```

FORTRAN77 Syntax

```
      integer*4 arg1(64),arg2(1), row

      do 100 row=0,7
         arg1(row*4+1) = Z'01010101';
         arg1(row*4+2) = Z'01010101';
         arg1(row*4+3) = 0;
         arg1(row*4+4) = 0;
100   continue
      do 200 row=8,15
         arg1(row*4+1) = 0;
         arg1(row*4+2) = 0;
         arg1(row*4+3) = Z'01010101';
         arg1(row*4+4) = Z'01010101';
200   continue

      call gescape(fildes,GR2D_FILL_PATTERN,arg1,arg2)
```

Pascal Syntax

```
{gescape_arg is defined in starbase.p1.h}

var
   arg1,arg2:gescape_arg;
   row: integer;
      :
begin
   for row := 0 to 7 do begin
      arg1.i[row*4+1] := hex('01010101');
      arg1.i[row*4+2] := hex('01010101');
      arg1.i[row*4+3] := 0;
      arg1.i[row*4+4] := 0;
   end;
   for row := 8 to 15 do begin
      arg1.i[row*4+1] := 0;
      arg1.i[row*4+2] := 0;
      arg1.i[row*4+3] := hex('01010101');
      arg1.i[row*4+4] := hex('01010101');
   end;

   gescape(fildes,GR2D_FILL_PATTERN,arg1,arg2);
```

GR2D_MASK_ENABLE

The ⟨*op*⟩ parameter is GR2D_MASK_ENABLE.

The HP 98548A, HP 98549A, HP 98550A, HP 319C displays, and the HP 98556A accelerator have hardware capability for 3-operand raster combination: that is, operations that use a tiling mask and a replacement rule that specify the combination of the source, mask, and destination. When enabled, the mask rule and current mask are used for `block_write` and `block_move` operations. When disabled, the normal replacement rule (see `drawing_mode`) is used and the current mask is ignored.

Related `gescapes` are GR2D_MASK_RULE and GR2D_DEF_MASK.

This `gescape` allows the user to enable or disable the use of the mask. The `arg1` parameter contains one flag. If `arg1[0]` is 0, 3-operand mode is disabled. If `arg1[0]` is 1, 3-operand mode is enabled for `block_write` and `block_move`. If `arg1[0]` is 2, 3-operand mode is enabled for `block_write`, `block_move` and raster text using the HP Window/9000 font manager or fast alpha libraries.

The `arg2` parameter is ignored.

This is a hardware-dependent feature not supported in window retained rasters.

C Syntax

```
/* gescape_arg is typedef defined in starbase.c.h */

gescape_arg arg1, arg2;
   .
   .
   .>
arg1.i[0]=TRUE;
gescape(fildes,GR2D_MASK_ENABLE,&arg1,&arg2);
```

FORTRAN77 Syntax

```
integer*4 arg1(64),arg2(64)
arg1(1)=TRUE
call gescape(fildes,GR2D_MASK_ENABLE,arg1,arg2)
```

Pascal Syntax

```
{gescape_arg is defined in starbase.p1.h}

var
arg1,arg2:gescape_arg;
  :
begin
arg1.i[1] := 1;
gescape(fildes,GR2D_MASK_ENABLE,arg1,arg2);
```

GR2D_MASK_RULE

The ⟨*op*⟩ parameter is `GR2D_MASK_RULE`.

The HP 98548A, HP 98549A, HP 98550A, and HP 319C displays, and the HP 98556A accelerator have hardware capability for three-operand raster combination: that is, operations that use a tiling mask and a replacement rule (drawing mode) that specify the combination of the source, mask, and destination. For more information, review the "Three-operand Raster Operations" section of the appropriate device driver chapter.

Related `gescapes` are `GR2D_MASK_ENABLE` and `GR2D_DEF_MASK`.

This `gescape` allows you to set the 3-operand drawing mode. The `arg1` parameter contains one integer, specifying the new replacement rule. The replacement rule is generated from the desired results by reading the eight result bits as a number. The default rule is ⟨*source*⟩, rule number `0xCC`.

Table GESC-3.

Mask	Source	Destination	Result
0	0	0	r0
0	0	1	r1
0	1	0	r2
0	1	1	r3
1	0	0	r4
1	0	1	r5
1	1	0	r6
1	1	1	r7

Table GESC-4.

bit→	7	6	5	4	3	2	1	0
rule→	r7	r6	r5	r4	r3	r2	r1	r0

For example, to derive the commonly used rule (if ⟨*mask*⟩ then ⟨*source*⟩ else ⟨*destination*⟩), rule number `0xCA`, the following table defines the rule. The rule number is determined by reading the result column as an integer (from bottom to top, r7 being the most significant bit and r0 the least significant).

Table GESC-5.

Mask	Source	Destination	Result
0	0	0	0
0	0	1	1
0	1	0	0
0	1	1	1
1	0	0	0
1	0	1	0
1	1	0	1
1	1	1	1

The arg2 parameter is ignored.

The following program fragment shows the use of this gescape and example rule.

C Syntax

```
/* gescape_arg is typedef defined in starbase.c.h */

gescape_arg arg1, arg2;
:
arg1.i[0]=0xCA;
gescape(fildes,GR2D_MASK_RULE,&arg1,&arg2);
```

FORTRAN77 Syntax

```
integer*4 arg1(64),arg2(64)
arg1(1)=Z'CA'
call gescape(fildes,GR2D_MASK_RULE,arg1,arg2)
```

Pascal Syntax

```
{gescape_arg is defined in starbase.p1.h}

var
arg1,arg2:gescape_arg;
  :
begin
arg1.i[1] := hex('CA');
gescape(fildes,GR2D_MASK_RULE,arg1,arg2);
```

GR2D_OVERLAY_TRANSPARENT

The ⟨*op*⟩ parameter is GR2D_OVERLAY_TRANSPARENT.

The HP 98549A, HP 98550A, and HP 319C displays, and the HP 98556A accelerator may be opened in configurations that provide 2-overlay planes, in addition to the 4- or-8 image planes. Images created in the overlay planes do not affect images in the graphics planes. However, pixels of value zero in the overlay planes may be made either transparent (allowing the graphics planes to be displayed) or opaque (obscuring the graphics planes).

This gescape allows the transparency of zero pixels to be turned on or off (the default is that zero pixels are transparent). The arg1 parameter contains a single flag: if TRUE, zero pixels are transparent; if FALSE, zero pixels are opaque and the color found in entry 0 of the overlay color map is displayed for those pixels.

The arg2 parameter is ignored.

This gescape should not be used with the HP 98548A display.

C Syntax

```
/* gescape_arg is typedef defined in starbase.c.h */

gescape_arg arg1, arg2;
:
arg1.i[0]=TRUE;
gescape(fildes,GR2D_OVERLAY_TRANSPARENT,&arg1,&arg2);
```

FORTRAN77 Syntax

```
integer*4 arg1(64),arg2(64)
arg1(1)=TRUE
call gescape(fildes,GR2D_OVERLAY_TRANSPARENT,arg1,arg2)
```

Pascal Syntax

```
{gescape_arg is defined in starbase.p1.h}

var
arg1,arg2:gescape_arg;
:
begin
arg1.i[1] := 1;
gescape(fildes,GR2D_OVERLAY_TRANSPARENT,arg1,arg2);
```

GR2D_PLANE_MASK

The ⟨*op*⟩ parameter is GR2D_PLANE_MASK.

This gescape defines a mask indicating the frame buffer planes read or written during bit/pixel block transfers. The mask is relevant when R_BIT_MODE has enabled bit/pixel mode and raw mode is used in block_read() and block_write(). The mask may define any number of planes up to the total number of planes opened. Extra bits are ignored. The least-significant bit in the mask corresponds to the least-significant accessible plane. For example, mask 5 allows reads and writes to both plane 0 and plane 2. The storage expected is that needed for the number of planes specified. For this example, storage for two planes is needed. Both planes are transferred on a single call to block_read or block_write. See the documentation of block_read and block_write with raw mode for more information.

This gescape overrides the mask set by gescape R_BIT_MASK. If this gescape is called after R_BIT_MASK, transfers to obscured regions of a retained raster (if supported) will be according to the most significant set bit in the mask value (i.e., only a single plane), and transfers to the visible regions will be according to the entire mask value. The R_BIT_MASK gescape must be used to ensure consistency in retained raster operations.

The arg1 parameter is the mask to be used.

The arg2 parameter is ignored.

The default mask is 0x01 (plane 0 only).

C Syntax

```
/* gescape_arg is typedef defined in starbase.c.h */

gescape_arg arg1, arg2;
.
.
.
arg1.i[0] = 5;
gescape(fildes,GR2D_PLANE_MASK,&arg1,&arg2);
```

FORTRAN77 Syntax

```
integer*4 arg1(64),arg2(64)
arg1.i(1) = 5
call gescape(fildes,GR2D_PLANE_MASK,arg1,arg2)
```

Pascal Syntax

```
{gescape_arg is defined in starbase.p1.h}

var
arg1,arg2 : gescape_arg;
  :
begin
arg1.i(1) := 5;
gescape(fildes,GR2D_PLANE_MASK,arg1,arg2);
```

GR2D_REPLICATE

The ⟨*op*⟩ parameter is `GR2D_REPLICATE`.

The HP 98548A, HP 98549A, HP 98550A, HP 319C displays, and the HP 98556A accelerator, have hardware support for pixel replication in the Y direction. This `gescape` combines driver management of replication in the X direction with the hardware support to provide square pixel replication, and specifies the replication factor to be used. The X replication needs a workspace, that must be specified in the call, along with the source and destination rectangles and the destination size. Only replication factors of 2, 4, 8, and 16 are supported.

The `arg1` parameter specifies the many parameters needed for the operation to take place:

`arg1[0]`	specifies replication factor. A value other than 2, 4, 8, or 16 is a no-op.
`arg1[1]`	specifies the source rectangle upper left X device coordinate.
`arg1[2]`	specifies the source rectangle upper left Y device coordinate.
`arg1[3]`	specifies the destination rectangle upper left X device coordinate.
`arg1[4]`	specifies the destination rectangle upper left Y device coordinate.
`arg1[5]`	specifies the destination rectangle X size in pixels.
`arg1[6]`	specifies the destination rectangle Y size in pixels.
`arg1[7]`	specifies the workspace upper left X device coordinate.
`arg1[8]`	specifies the workspace upper left Y device coordinate.
`arg1[9]`	specifies the workspace X size in pixels.
`arg1[10]`	specifies the workspace Y size in pixels.
`arg1[11]`	specifies whether the workspace position has been specified in window device coordinates or raw device coordinates: (0 = window coordinates, 1 = raw coordinates)

The `arg2` parameter is ignored.

When `arg1` is an allowed value, replication is done as follows:

- The number of source pixels to be replicated is computed based on the source, workspace, and destination sizes (one of them being the limiting factor).
- The workspace is cleared to zeroes.
- The X-replication is done in the workspace.
- The driver waits for a vertical retrace.
- The final Y-replication is done to the destination rectangle.

Some notes on the use of this `gescape`:

- For the workspace not to be the limiting component in either the X or Y dimension, it must be as wide as the destination in X, and as high as the source in Y.
- The workspace may overlap the destination rectangle only at the bottom of the destination.
- If the workspace is on-screen and visible, visually displeasing effects may occur during the X-replication. Usually, it is desirable to either use an offscreen workspace (acquired with the `gescapes` `R_OFFSCREEN_ALLOC` or `R_FULL_FRAME_BUFFER`), or to blank out the workspace by obscuring it with a mask in another image plane or an overlay plane.
- Large destination areas may appear to "tear" during the replication because a video refresh occurs during the final replication operation. The largest destination that may safely be used without risk of tearing is 512×512 pixels.
- This is a hardware-dependent feature and is not supported in window retained rasters.

The following example replicates a 100×100 source at X=0,Y=0 into a 400×400 destination at X=512,y=0 using a on-screen workspace of minimum size at X=512,Y=512.

C Syntax

```
/* gescape_arg is typedef defined in starbase.c.h */

gescape_arg arg1, arg2;
:
arg1.i[0]=4;
arg1.i[1]=0;
arg1.i[2]=0;
arg1.i[3]=512;
arg1.i[4]=0;
arg1.i[5]=400;
arg1.i[6]=400;
arg1.i[7]=512;
arg1.i[8]=512;
arg1.i[9]=400;
arg1.i[10]=100;
arg1.i[11]=0;
gescape(fildes, GR2D_REPLICATE, &arg1, &arg2);
```

FORTRAN77 Syntax

```
integer*4 arg1(64),arg2(64)
arg1(1)=4;
arg1(2)=0;
arg1(3)=0;
arg1(4)=512;
arg1(5)=0;
arg1(6)=400;
arg1(7)=400;
arg1(8)=512;
arg1(9)=512;
arg1(10)=400;
arg1(11)=100;
arg1(12)=0;
call gescape(fildes, GR2D_REPLICATE, arg1, arg2)
```

Pascal Syntax

```
{gescape_arg is defined in starbase.p1.h}

var
arg1,arg2:gescape_arg;
  :
begin
arg1.i[1] := 4;
arg1.i[2] := 0;
arg1.i[3] := 0;
arg1.i[4] := 512;
arg1.i[5] := 0;
arg1.i[6] := 400;
arg1.i[7] := 400;
arg1.i[8] := 512;
arg1.i[9] := 512;
arg1.i[10] := 400;
arg1.i[11] := 100;
arg1.i[12] := 0;
gescape(fildes, GR2D_REPLICATE, arg1, arg2);
```

HPGL_READ_BUFFER

The ⟨*op*⟩ parameter is HPGL_READ_BUFFER.

This gescape allows you to read data from the device. The arg1 parameter is a character buffer that the device string will be returned in. The arg2 parameter is the length of the string in bytes.

This gescape assumes the user program has sent an output command to the device previous to this call, possibly using the HPGL_WRITE_BUFFER gescape. The device driver will flush the current output buffer, send a serial trigger to the device if necessary, and then read in the device's reply.

Caution If the user program has *not* sent an HP-GL output command OI* before this gescape, the application program will wait indefinitely for a reply when no timeout is set.

C Syntax Example

```
    /* gescape_arg is a type defined in starbase.c.h */
gescape_arg arg1, arg2;

    /* First we send the HP-GL output command for its ID */
strcpy(arg1.c, "OI;");
arg2.i[0] = 3;
gescape(fildes, HPGL_WRITE_BUFFER, &arg1, &arg2);

    /* Now we read back in the device's ID */
gescape(fildes, HPGL_READ_BUFFER, &arg1, &arg2);
printf("The ID is %s and has %d letters in it", arg1.c, arg2.i[0]);
```

FORTRAN77 Syntax Example

```
C
        CHARACTER ARG1C(255)
        INTEGER ARG2I(64)
C
C FIRST WE SEND THE HP-GL OUTPUT COMMAND FOR ITS ID
C
        ARG1C(1) = 'O'
 ARG1C(2) = 'I'
        ARG1C(3) = ';'
        ARG2I(1) = 3
        CALL GESCAPE(FILDES, HPGL_WRITE_BUFFER, ARG1C, ARG2I)
C
C NOW WE READ BACK IN THE DEVICE'S ID
C
        CALL GESCAPE(FILDES, HPGL_READ_BUFFER, ARG1C, ARG2I)
        PRINT*, 'THE ID IS ', ARG1C, 'AND HAS ',ARG2I(1),
        'LETTERS IN IT'
```

Pascal Syntax Example

```
    { gescape_arg is defined in starbase.p1.h }
var
    arg1, arg2 : gescape_arg;
begin
        { First we send the HP-GL output command for its ID }
    arg1.c := 'OI;';
    arg2.i[1] := 3;
    gescape(fildes, HPGL_WRITE_BUFFER, arg1, arg2);

        { Now we read back in the device's ID }
    gescape(fildes, HPGL_READ_BUFFER, arg1, arg2);
    writeln('The ID is ', arg1.c, ' and has ',
       arg2.i[1], 'letters in it');
```

HPGL_SET_PEN_NUM

The ⟨*op*⟩ parameter is HPGL_SET_PEN_NUM.

This gescape allows you to explicitly state the number of pens.

The arg1 parameter is the number of pens.

The arg2 parameter is ignored.

The following examples change the number of pens to 6.

C Syntax Example

```
/* gescape_arg is type defined in starbase.c.h */
gescape_arg arg1, arg2;
arg1.i[0] = 6;
gescape(fildes, HPGL_SET_PEN_NUM, &arg1, &arg2);
```

FORTRAN77 Syntax Example

```
integer arg1i(64), arg2i(64)
arg1i(1) = 6
call gescape(fildes, HPGL_SET_PEN_NUM, arg1i, arg2i)
```

Pascal Syntax Example

```
{gescape_arg is defined in starbase.p1.h}
var
arg1,arg2 : gescape_arg;
begin
arg1.i[1] := 6;
gescape(fildes, HPGL_SET_PEN_NUM, arg1, arg2);
end.
```

When you change the number of pens, a new color map of the appropriate size is created and initialized to the Starbase default color map entries. The size is number_of_pens+1 (the extra one is for pen up).

HPGL_SET_PEN_SPEED

The ⟨*op*⟩ parameter is HPGL_SET_PEN_SPEED.

This gescape allows you to change pen velocity.

The arg1 parameter is the desired pen velocity. Pen velocity is specified in centimeters per second.

The arg2 parameter may be 0 (zero) to specify the new velocity for all pens or set to the specific pen number to have that pen's velocity changed. If the desired velocity is out of range for the device, the result is device dependent. If the pen number is out of range, the result is also device dependent.

The following example will set the the velocity to 30 centimeters per second for all pens.

C Syntax Example

```
/* gescape_arg is type defined in starbase.c.h */
gescape_arg arg1, arg2;
arg1.i[0] = 30;
arg2.i[0] = 0;
gescape(fildes, HPGL_SET_PEN_SPEED, &arg1, &arg2);
```

FORTRAN77 Syntax Example

```
integer arg1i(64), arg2i(64)
arg1i(1) = 30
arg2i(1) = 0
call gescape(fildes, HPGL_SET_PEN_SPEED, arg1i, arg2i)
```

Pascal Syntax Example

```
{gescape_arg is defined in starbase.p1.h}
var
arg1,arg2 : gescape_arg;
begin
arg1.i[1] := 30;
arg2.i[1] := 0;
gescape(fildes, HPGL_SET_PEN_SPEED, arg1, arg2);
end.
```

HPGL_SET_PEN_WIDTH

The ⟨*op*⟩ parameter is HPGL_SET_PEN_WIDTH.

This gescape allows a change in pen width.

The arg1 parameter is the desired pen width. Pen width is specified in millimeters (mm).

The arg2 parameter is the distance between fill lines in millimeters.

The following example sets the pen width to 0.4 millimeters and the distance between fill lines to 0.6 millimeters. Pen width is used in calculating the distance between lines when performing area fill.

C Syntax Example

```
/* gescape_arg is type defined in starbase.c.h */
gescape_arg  arg1, arg2;
arg1.f[0] = 0.4;
arg2.f[0] = 0.6;
gescape(fildes, HPGL_SET_PEN_WIDTH, &arg1, &arg2);
```

FORTRAN77 Syntax Example

```
real arg1f(64), arg2f(64)
arg1f(1) = 0.4
arg2f(1) = 0.6
call gescape(fildes, HPGL_SET_PEN_WIDTH, arg1f, arg2f)
```

Pascal Syntax Example

```
{gescape_arg is defined in starbase.p1.h}
var
arg1,arg2 : gescape_arg;
begin
arg1.f[1] := 0.4;
arg2.f[1] := 0.6;
gescape(fildes, HPGL_SET_PEN_WIDTH, arg1, arg2);
end.
```

HPGL_WRITE_BUFFER

The ⟨*op*⟩ parameter is HPGL_WRITE_BUFFER.

This gescape permits direct communication of HP-GL commands to supported devices. The commands are sent directly to the device without alteration. Invalid commands will cause unpredictable results. The full HP-GL command syntax must be observed, including proper placement of punctuation.

The arg1 parameter is an ASCII buffer of HP-GL commands with a maximum length of 255 bytes.

The arg2 parameter is the command buffer's length in bytes.

C Syntax Example

```
/* gescape_arg is type defined in starbase.c.h */
gescape_arg arg1, arg2;
strcpy (arg1.c,"PU;");
arg2.i[0] = 3;
gescape(fildes, HPGL_WRITE_BUFFER, &arg1, &arg2);
```

FORTRAN77 Syntax Example

```
character arg1c(255)
integer arg2i(64)
arg1c(1) = 'P'
arg1c(2) = 'U'
arg1c(3) = ';'
arg2i(1) =   3
call gescape(fildes, HPGL_WRITE_BUFFER, arg1c, arg2i)
```

Pascal Syntax Example

```
{gescape_arg is defined in starbase.p1.h}
var
arg1,arg2 : gescape_arg;
begin
arg1.c[1] := 'P';
arg1.c[2] := 'U';
arg1.c[3] := ';';
arg2.i[1] := 3;
gescape(fildes, HPGL_WRITE_BUFFER, arg1, arg2);
end.
```

IGNORE_RELEASE

The ⟨*op*⟩ parameter is IGNORE_RELEASE.

This gescape causes the event trigger to start only when a button is pressed. This reverses the condition created by TRIGGER_ON_RELEASE.

This is the default condition.

The arg1 and arg2 parameters are ignored.

C Syntax Example

```
/* gescape_arg is type defined in starbase.c.h */
gescape_arg arg1, arg2;
gescape(fildes,IGNORE_RELEASE,&arg1,&arg2);
```

FORTRAN77 Syntax Example

```
integer*4 arg1(64),arg2(64)
call gescape(fildes,IGNORE_RELEASE,arg1,arg2)
```

Pascal Syntax Example

```
{gescape_arg is defined in starbase.p1.h}
var
   arg1, arg2 : gescape_arg;
begin
   gescape(fildes,IGNORE_RELEASE,arg1,arg2);
```

IMAGE_BLEND

The ⟨*op*⟩ parameter is `IMAGE_BLEND`.

The HP 98730 and HP 98731 Device Drivers support video blending hardware.

This `gescape` allows the user to enable or disable video blending of frame buffer banks.

The `arg1` points to a flag which enables blending if equal to one and disables blending if zero. When this `gescape` is called, the program must be `gopened` to the image planes, and double buffering must be turned off. Double buffering may be turned on after blending is enabled.

The `arg2` is ignored.

This `gescape` command will override any previous display modes set by `shade_mode` or `double_buffering`. Subsequent calls to `bank_switch` can be used to alter the frame buffer banks being written and displayed, and `shade_mode` can be used to initialize the color table and set the color map mode within the banks.

When blending is enabled, the hardware configuration is altered in the following way:

- While blending is enabled, `bank_switch` may be used to select single or multiple banks to be displayed. During blending, the `dbank` parameter is used as a mask with bit 0 corresponding to bank 0, bit one corresponding to bank 1, and so forth. Only one bank at a time may be selected for writing.
- Any combination of the three available frame buffer banks may be displayed simultaneously. Each frame buffer bank of 8 display planes has its own 256 entry color map. These color maps may be individually set using call to the `SET_BANK_CMAP gescape` and regular `define_color_table` calls. See the `SET_BANK_CMAP` documentation for details. By default all three color maps contain the same entries, which were specified with the last `define_color_table` or `shade_mode` call. When multiple frame buffers are turned on, the intensities out of the separate color maps are summed and displayed. if the sum for any red, green, or blue color exceeds the maximum allowable intensity of 1.0, it is clamped.

- Display enable and `write_enable` become 24 bit quantities, with one bit for each of the 24 display planes which can be installed. (See `display_enable` and `write_enable`.) When blending is turned on, the lower byte of the plane is duplicated into the next two upper bytes to get 24 bit quantities. Subsequent calls to `display_enable` and `write_enable` will treat the enable value as 24 bit quantities.
- Since each display bank has its own 256 entry color map, 24 bit color is not available when blending. If the color map mode is set to `CMAP_FULL` while blending is enabled, eight planes will be used, with three bits for red, three bits for green, and two bits for blue.
- The blink mask given to the `gescape` `BLINK_PLANES` becomes a 24 bit quantity, with one bit for each of the 24 display planes which can be installed. (See `BLINK_PLANES`.) If blinking of planes is currently enabled when blending is turned on, the byte blink mask specified previously by the user is duplicated into the next two upper bytes to obtain a 24 bit blink planes mask.
- Since all the hardware color maps are used for blending when it is enabled, blinking color map entries are not supported simultaneously with blending. This means that calls to `gescape` with an ⟨*op*⟩ parameter of `BLINK_INDEX` will have no effect (although `BLINK_PLANES` still works—see above).

The following examples demonstrate how to use this function.

C Syntax

```
/* gescape_arg is typedef defined in starbase.c.h */

gescape_arg arg1, arg2;
   :
arg1.i[0] = 1;
gescape(fildes,IMAGE_BLEND,&arg1,&arg2);
bank_switch(fildes,0,7);  /* enable blending of all three banks */
   :
arg1.i =0;  /* disable blending */
gescape(fildes,IMAGE_BLEND,&arg1,&arg2);
bank_switch(fildes,0,0);   /* return to normal operation */
```

FORTRAN77 Syntax

```
integer*4 arg1(4),arg2(1)

arg1(1)=1
call gescape(fildes,IMAGE_BLEND,arg1,arg2)
      call bank_switch(fildes,0,7)
  :
arg1(1)=0
call gescape(fildes,IMAGE_BLEND,arg1,arg2)
      call bank_switch(fildes,0,0)
```

Pascal Syntax

```
{gescape_arg is defined in starbase.p1.h}

var
arg1,arg2:gescape_arg;
   :
begin
arg1.i[1]:= 1;
gescape(fildes,IMAGE_BLEND,arg1,arg2);
  bank_switch(fildes,0,0);
     :
arg1.i[1]:= 0;
gescape(fildes,IMAGE_BLEND,arg1,arg2);
  bank_switch(fildes,0,0);
end
```

LS_OVERFLOW_CONTROL

The ⟨*op*⟩ parameter is `LS_OVERFLOW_CONTROL`.

Four options are provided to the user to address overflow situations that may occur with light source calculations. This `gescape` takes up to 4 floating point numbers in `arg1` to set the option.

`CLIPPED` This is the default option and will be the fastest case. The red, green, and blue values are calculated, and if any color value exceeds 1.0 it is truncated to 1.0. (`r=min(r,1.0,)`, etc.)

To get this option the `gescape` should be called with: `arg1.f[0]=0.0`

`SCALED` When an overflow occurs, this option maintains the proper hue. If any color exceeds 1.0, the maximum of the red, green, and blue values is used to divide each of the color values with. (`r=r/max(r,g,b)`, etc.)

To get this option the gescape should be called with: `arg1.f[0]=1.0`

`DEBUG` This option allows the user to quickly determine where the light source equations are overflowing. The user selects a color and then if any overflow occurs the overflow color is used instead of the calculated color.

To get this option, the `gescape` should be called with:
`arg1.f[0]=3.0`,
`arg1.f[1]=`⟨*red overflow component*⟩,
`arg1.f[2]=`⟨*green overflow component*⟩,
`arg1.f[3]=`⟨*blue overflow component*⟩.

HP 98721 Only

`HYBRID` This option scales the diffuse and specular terms separately, then multiplies the diffuse term by a fractional value and adds it to the specular term. This new color is clipped if necessary.

(`r=min((rd/max(rd,gd,bd))*diff+rs/max(rs,gs,bs),1.0)`, etc)

This option allows the user to limit the diffuse term to some fraction of the full color and the specular contribution will bring it to full color. The assurance that the diffuse is in a certain region and the

specular is in another region is useful for two reasons. When in CMAP_FULL mode it guarantees that the specular reflections can be seen. When in CMAP_MONOTONIC mode the color map can be set up in such a way to have the diffuse color in one region and the specular color in the next region of the color map.

To get this option the gescape should be called with:
arg1.f[0]=2.0,
arg1[1]=⟨*diffuse fraction*⟩
The value of ⟨*diffuse fraction*⟩ is 0.0 to 1.0.

When using a HP 98730 device, similar functionality to the HYBRID option is available with surface_coefficients.

The following examples select a scaled model.

C Syntax

```
/*gescape_arg is typedef defined in starbase.c.h */

gescape_arg arg1, arg2;
   ⋮
arg1.f[0]=1.0;
gescape(fildes,LS_OVERFLOW_CONTROL,&arg1,&arg2);
```

FORTRAN77 Syntax

```
real arg1(64),arg2(64)
arg1(1)=1.0
call gescape(fildes,LS_OVERFLOW_CONTROL,arg1,arg2)
```

Pascal Syntax

```
{gescape_arg is defined in starbase.p1.h}

var
   arg1,arg2:gescape_arg;
       ⋮
begin
   arg1.f[1] := 1.0;
   gescape(fildes,LS_OVERFLOW_CONTROL,arg1,arg2);
```

OVERLAY_BLEND

The ⟨*op*⟩ parameter is `OVERLAY_BLEND`.

The HP 98730 and HP 98731 overlay color map defines a transparency bit associated with a color map entry. If the transparency bit for a pixel is set to one, the pixel color is forced to the red, green, and blue values in the overlay color map. If the transparency bit is set to zero, the red, green, and blue values in the overlay planes are blended with the red, green, and blue values in the graphics planes behind the overlay planes. If the red, green, and blue values for the selected entry are all zero, then black will be blended with the graphics planes. Blending black is exactly the same as defining an entry to be transparent.

This `gescape` lets you control the blending of overlay color map transparent entries.

`arg1[0]` is an index value in the range of 0–15 (only 0–7 if the overlay planes were opened with a 3-plane device file) defining which color map entry for which the transparency mode is being defined. If the value is not in the range of 0–15 a mod function is performed.

If `arg1[1]` is `TRUE`, writing a pixel with the specified index value results in blending the color defined for that entry in the overlay planes with the graphics planes.

If `arg1[1]` is `FALSE`, writing a pixel with the specified index value results in black being blended with the graphics planes.

For processes in the image planes using the fourth overlay plane for cursors (see the `R_OVERLAY_ECHO gescape`), the cursor will not be visible in regions of transparency where black is not being blended with the image planes.

The `arg2` parameter is ignored.

C Syntax

```
/*gescape_arg is typedef defined in starbase.c.h */

gescape_arg arg1, arg2;
   :
arg1.i[0]=3;
arg1.i[1]=FALSE;
gescape(fildes,OVERLAY_BLEND,&arg1,&arg2);
```

FORTRAN77 Syntax

```
integer*4 arg1(64), arg2(64)
arg1(1)=3
arg1(2)=0
call gescape(fildes,OVERLAY_BLEND,arg1,arg2)
```

Pascal Syntax

```
{gescape_arg is defined in starbase.p1.h}

var
   arg1,arg2:gescape_arg;
   :
begin
   arg1.i[1] := 3;
   arg1.i[2] := FALSE;
   gescape(fildes,OVERLAY_BLEND,arg1,arg2);
```

PAN_AND_ZOOM

The ⟨*op*⟩ parameter is `PAN_AND_ZOOM`.

The `PAN_AND_ZOOM gescape` is only supported in image planes.

Pixel panning is a function that readdresses the start of the video scan to arbitrary X, Y pixels.

Pixel zooming is a function most commonly used with pixel panning to inspect pixels in an image by enlarging the image. Zooming is a replication of pixels.

For example: a 2X zoom replicates each pixel four times, twice in the X direction, and twice in the Y direction.

Pixel pan and zoom can be done relative to the upper left of the screen, or relative to the center of the screen.

This `gescape` can be used with the HP 98730 and HP 98731 device drivers to control pixel pan and zoom hardware.

If `arg1[0]` is `TRUE` (1) then `arg1[1]` is the X location (call it PAN_X), and `arg1[2]` is the Y location (call it PAN_Y), of the frame buffer pixel to be at the center of the screen.

If `arg1[0]` is `FALSE` (0) then `arg1[1]` is the X location (call it $PAN_{X)}$, and `arg1[2]` is the Y location (call it PAN_Y), of the frame buffer pixel to be at the upper left-most position of the screen.

Resolution in PAN_X is limited to four pixel boundaries. This means that the lower two bits of the value passed in for PAN_X are masked off.

`arg1[3]` is the zoom factor for pixel replication. Legal values are values from 1–16. A zoom factor of one or zero implies no pixel replication.

`arg1[4]` is either `TRUE` (1) or `FALSE` (0). Setting this parameter to `TRUE` allows PAN_X values such that pixels beyond 2047 are displayed on the screen. Setting this parameter to `FALSE` results in PAN_X values being adjusted so that no wrap around is attempted.

Wrap around can occur in the Y direction. Wrap around cannot occur in the X direction. Therefore, pixels beyond 2047 are undefined.

If PAN_X or PAN_Y are negative values, then they are converted to positive values by adding either 2048 or 1024 respectively, until the value becomes positive.

Setting `arg1[0]` to `FALSE` (0), `arg1[1]` and `arg1[2]` to 0, and `arg1[3]` to 1 will obtain normal displaying, where pixel 0,0 is the upper-left screen origin and no pixel replication is done.

At gopen time the pixel pan and zoom hardware is reset to obtain normal display if the Mode word contains `INIT` or `RESET_DEVICE`. Otherwise, the pixel pan and zoom hardware is left in its current state.

C Syntax

```
/*gescape_arg is typedef defined in starbase.c.h */

   gescape_arg arg1, arg2;
      ⋮
   arg1.i[0]=1;  /* Request that PANX,PANY represent pixel to become center
*/
               /* of the screen */
   arg1.i[1]=1280;
   arg1.i[2]=512;
   arg1.i[3]=1;     /* Do not do any pixel replication */
   arg1.i[4]=0;
   gescape(fildes,PAN_AND_ZOOM,&arg1,&arg2);
```

FORTRAN77 Syntax

```
integer*4 arg1(64),arg2(64)
arg1(1)=1
arg1(2)=1280
arg1(3)=512
arg1(4)=1
arg1(5)=0
call gescape(fildes,PAN_AND_ZOOM,arg1,arg2)
```

Pascal Syntax

```
{gescape_arg is defined in starbase.p1.h}

var
   arg1,arg2:gescape_arg;
      :
begin
   arg1.i[1] := 1;
   arg1.i[2] := 1280;
   arg1.i[3] := 512;
   arg1.i[4] := 1;
   arg1.i[5] := 0;
   gescape(fildes,PAN_AND_ZOOM,arg1,arg2);
```

PATTERN_FILL

The ⟨*op*⟩ parameter is `PATTERN_FILL`.

This `gescape` allows you to fill polygons with a pattern stored in offscreen memory. Shade mode must be `CMAP_FULL` or `CMAP_MONOTONIC` for this `gescape` to work. To resume non-pattern operations call `drawing_mode` to change the replacement rule. Refer to the *Starbase Graphics Techniques* manual for more information on replacement rules.

Using this `gescape`, you specify one of 256 replacement rules that include a pattern. The new replacement rule is hex number obtained from using a logical operator on three inputs:

⟨*pattern*⟩ *op* ⟨*source*⟩ *op* ⟨*destination*⟩ → ⟨*result*⟩

A ⟨*pattern*⟩ is a rectangular grid of off-screen pixels containing a pattern you will use to "overlay" with the ⟨*source*⟩ and and ⟨*destination*⟩ information. You are sending the new information, ⟨*source*⟩, to the pixel. The information currently in the pixel is ⟨*destination*⟩.

The replacement rule is used to determine how data is written into the frame buffer. Since there are eight possible ways to combine three-operands, in this case the ⟨*source*⟩, ⟨*destination*⟩, and ⟨*pattern*⟩, there are eight bits in the replacement rule. The following table shows the bit from the replacement rule which will be used for each of the logical combinations.

Table GESC-6. Replacement Rule Truth Table

Pattern	Source	Destination	Result Bits
0	0	0	r7
0	0	1	r6
0	1	0	r5
0	1	1	r4
1	0	0	r3
1	0	1	r2
1	1	0	r1
1	1	1	r0

Note that if you wish to duplicate the effect of `drawing_mode`'s replacement rules, copy the values of r3-r0 into this rule's r7-r4 and r3-r0. This makes the ⟨*pattern*⟩ operand a no-op.

The following table shows five example replacement rules.

Table GESC-7. Example Replacement Rules

Rule, and Hex Representation of the Rule	r7	r6	r5	r4	r3	r2	r1	r0
Zero 0x00	0	0	0	0	0	0	0	0
Source 0x33	0	0	1	1	0	0	1	1
Source OR Destination 0x77	0	1	1	1	0	1	1	1
If pattern=0,then Destination If pattern=1,then Source 0x53	0	1	0	1	0	0	1	1
Pattern 0x0F	0	0	0	0	1	1	1	1

The first three examples are replacements that can also be done using `drawing_mode`: the upper four bits of the replacement rule are the same as the lower four bits and match the equivalent `drawing_mode` rule. The last two examples are replacement rules that use the ⟨*Pattern*⟩ operand. In the fourth example, when pattern bit is 0, the result bit remains unchanged; when the pattern bit is 1, the result bit is set equal to the source. In the last example, the result bit is equal to the pattern bit.

The `gescape` takes five parameters:

- The first parameter is one of the replacement rules described above.
- The second parameter is the X location of the upper left corner of the pattern rectangle.
- The third parameter is the Y location of the upper left corner of the pattern rectangle.

- The fourth parameter is the dx size of the pattern rectangle. allowable values of dx are 16, 32, 64, 128, 256. x mod dx must equal zero.
- The fifth parameter is the dy size of the pattern rectangle. Allowable values of dy are 4, 8, 16, 32, 64, 128, 256. y mod dy must equal zero.

The following example shows a pattern in the upper right corner of off-screen memory. The pattern is 128×128 and is located at (1920, 0). Subsequent polygon primitives and vector primitives will use the pattern until **drawing_mode** is called.

C Syntax

```
/*gescape_arg is typedef defined in starbase.c.h */

   gescape_arg arg1, arg2;
   char patt[128][128];
      :
   dcblock_write(fildes,1920,0,128,128,patt,1);
   arg1.i[0]=0x0F;
   arg1.i[1]=1920;
   arg1.i[2]=0;
   arg1.i[3]=128;
   arg1.i[4]=128;
   gescape(fildes,PATTERN_FILL,&arg1,&arg2);
   rectangle(fildes,x1,y1,x2,y2);
   drawing_mode(fildes,3);
```

FORTRAN77 Syntax

```
integer arg1(64),arg2(64)
integer patt(32,32)

call dcblock_write(fildes,1920,0,128,128,patt,1)
arg1(1)=15
arg1(2)=1920
arg1(3)=0
arg1(4)=128
arg1(5)=128
call gescape(fildes,PATTERN_FILL,arg1,arg2)
call rectangle(fildes,x1,y1,x2,y2)
call drawing_mode(fildes,3)
```

Pascal Syntax

```
{gescape_arg is defined in starbase.p1.h}

var
   arg1,arg2:gescape_arg;
       patt:array [0..127,0..127] of 0..255;
       :
begin
       dcblock_write(fildes,1920,0,128,128,patt,1);
       arg1.i[0] :=hex('0F');
       arg1.i[1] :=1920;
       arg1.i[2] :=0;
       arg1.i[3] :=128;
       arg1.i[4] :=128;
   gescape(fildes,PATTERN_FILL,arg1,arg2);
       rectangle(fildes,x1,y1,x2,y2);
       drawing_mode(fildes,3);
```

R_BIT_MASK

The ⟨*op*⟩ parameter is R_BIT_MASK.

This gescape defines a mask. The mask indicates the plane to read bit patterns from or write bit patterns to. The highest plane indicated by the mask is the enabled plane. For example, mask 5 allows reads and writes to plane 2.

The arg1 parameter defines the mask to be used. The range of values allowed for this parameter are device dependent. The default mask is 1.

The arg2 parameter is ignored.

Note This gescape should not be used on black and white devices.

C Syntax

```
/* gescape_arg is typedef defined in starbase.c.h */

gescape_arg arg1, arg2;
   :
arg1.i[0] = 5;
gescape(fildes,R_BIT_MASK,&arg1,&arg2);
```

FORTRAN77 Syntax

```
 integer*4 arg1(64),arg2(64)
 arg1(1) = 5
 call gescape(fildes,R_BIT_MASK,arg1,arg2)
```

Pascal Syntax

```
{gescape_arg is defined in starbase.p1.h}

var
   arg1,arg2 : gescape_arg;
   :
begin
   arg1.i[1] := 5;
   gescape(fildes,R_BIT_MASK,arg1,arg2);
```

R_BIT_MODE

The ⟨*op*⟩ parameter is R_BIT_MODE.

This gescape changes the definition of the raw mode flag for block reads and writes. When bit_mode is turned on (arg1 is true) and a block read or write is called with the raw mode flag set (true), then each byte of the source/destination array contains information about eight pixels, 1-bit per pixel. The plane read from or written into is set using the gescape R_BIT_MASK. When in "bit mode", raw reads and writes are clipped. Each row of the bit pattern must be padded to a byte boundary. For Example:

If a block write with a "destination x" of 18 bits is performed, each row of the bit pattern is three bytes long. The first bit (highest order bit) of the first byte of the source bit pattern, will determine the first pixel on the destination device. The second bit of the first byte determines the second pixel, and so on through the first two bytes. The 17th pixel is determined by the first bit of the third byte, while the last pixel on the first row of the device is determined by the second bit of the third byte. The next six bits of byte three are ignored. Then the first pixel of the second row is represented by the first bit of the fourth byte.

A bit pattern that turns on every third pixel in each row of an 18×2 pixel area would look like this (each digit represents a single bit and the spaces represent byte boundaries).

```
00100100 10010010 01000000
00100100 10010010 01000000
```

"Bit mode" can be used to reduce the amount of space it takes to store a raster image.

If arg1 is TRUE (1), raw mode is 1 bit per pixel. If arg1 if FALSE (0), raw mode is used.

The arg2 parameter is ignored.

C Syntax

```
/* gescape_arg is typedef defined in starbase.c.h */

gescape_arg arg1, arg2;
   :
arg1.i[0] = 1;
gescape(fildes,R_BIT_MODE,&arg1,&arg2);
```

FORTRAN77 Syntax

```
integer*4 arg1(64),arg2(64)
arg1(1) = 1
call gescape(fildes,R_BIT_MODE,arg1,arg2)
```

Pascal Syntax

```
{gescape_arg is defined in starbase.p1.h}

var
   arg1,arg2 : gescape_arg;
   :
begin
   arg1.i[1] := 1;
   gescape(fildes,R_BIT_MODE,arg1,arg2);
```

R_DEF_ECHO_TRANS

The ⟨*op*⟩ parameter is `R_DEF_ECHO_TRANS`.

This `gescape` allows you to define a transparency mask for raster cursors. The transparency mask is used to determine which bits of the raster cursor pattern are visible over the graphics background. The transparency mask is assumed to have the same height and width as the current raster cursor. The mask is arranged in a packed array as one-bit per pixel, so each byte contains information about eight pixels. If the bit in the mask is set, the corresponding pixel location in the raster cursor will be visible. If the pattern bit is zero, the corresponding pixel will be transparent (not drawn).

This `gescape` provides the same functionality as `R_ECHO_MASK`, except that input data to `R_ECHO_MASK` is byte aligned on row boundaries. Suggestion: On the HP 98730 and HP 98731 devices, use `R_ECHO_MASK` for slightly better performance.

With the HP 98730 Device Driver, echo transparency patterns cannot be used in graphics windows if the echo currently being used is not the hardware cursor, or the echos are not overlayed in the fourth overlay plane.

After this `gescape` has been called, the transparency mask will be used to draw the current raster cursor, until another raster cursor is defined with a call to `define_raster_echo`. If `define_raster_echo` is called, it is necessary to follow that call with another call to this `gescape` to use a transparency mask. To summarize, calling this `gescape` turns on transparency, calling `define_raster_echo` turns off transparency.

Note Because of hardware limitations, only a transparency mask size up to 16×16 pixels is supported by `hp98550` and `hp98556` device drivers. If the current raster echo has a larger size, this `gescape` will have no effect.

The `arg1` parameter is assumed to point to the transparency mask.

The `arg2` parameter is ignored.

The following program segments show how to define a transparency mask for the default raster cursor.

C Syntax

```
/* gescape_arg is typedef defined in starbase.c.h */

gescape_arg arg1, arg2;
   :
arg1.i[0]=0xF0C0A090;
arg1.i[1]=0x08040201;
gescape(fildes,R_DEF_ECHO_TRANS,&arg1,&arg2);
```

FORTRAN77 Syntax

```
integer*4 arg1(64),arg2(64)
arg1(1)=Z'F0C0A090'
arg1(2)=Z'08040201'
call gescape(fildes,R_DEF_ECHO_TRANS,arg1,arg2)
```

Pascal Syntax

```
{gescape_arg is defined in starbase.p1.h}

var
   arg1,arg2:gescape_arg;
   :
begin
   arg1.i[1] := hex('F0C0A090');
   arg1.i[2] := hex('08040201');
   gescape(fildes,R_DEF_ECHO_TRANS,arg1,arg2,null);
```

R_DEF_FILL_PAT

The ⟨*op*⟩ parameter is R_DEF_FILL_PAT.

This gescape allows the user to specifically define the current 4×4 pixel dither cell when in CMAP_NORMAL color map mode. This gescape will not work in CMAP_MONOTONIC or CMAP_FULL color map modes. See the shade_mode information in the *Starbase Reference*. The dither cell is used to fill polygon and rectangle primitives. Suggestion: Use the INT_PATTERN interior style instead of this gescape.

The arg1 parameter specifies the bytes defining the dither cell. The 16 bytes are placed in the dither cell in row major order. After gescape is called, the polygon and rectangle primitives will be filled with the user-defined pattern until another pattern is defined with another gescape call or until fill_color is called.

The arg2 parameter is ignored.

C Syntax

```
/* gescape_arg is typedef defined in starbase.c.h */

gescape_arg arg1, arg2;
   :
arg1.c[0] =1; arg1.c[1] =0; arg1.c[2] =0; arg1.c[3] =0;
arg1.c[4] =0; arg1.c[5] =1; arg1.c[6] =0; arg1.c[7] =0;
arg1.c[8] =0; arg1.c[9] =0; arg1.c[10]=1; arg1.c[11]=0;
arg1.c[12]=0; arg1.c[13]=0; arg1.c[14]=0; arg1.c[15]=1;

gescape(fildes,R_DEF_FILL_PAT,&arg1,&arg2);
```

FORTRAN77 Syntax

```
      integer*4 arg1(64),arg2(64),pattern(4)
      data arg1/z'01000000',
     C          z/00010000',
     C          z/00000100',
     C          z/00000001'/
      call gescape(fildes,R_DEF_FILL_PAT,arg1,arg2)
```

Pascal Syntax

```
{gescape_arg is defined in starbase.p1.h}

var
   arg1,arg2:gescape_arg;
      .
      .
      .
begin
   arg1.i[1] := hex('01000000');
   arg1.i[2] := hex('00010000');
   arg1.i[3] := hex('00000100');
   arg1.i[4] := hex('00000001');
   gescape(fildes,R_DEF_FILL_PAT,arg1,arg2);
```

R_DMA_MODE

The ⟨*op*⟩ parameter is `R_DMA_MODE`.

This `gescape` changes the definition of the `raw` flag for block writes. When DMA mode is turned on (`arg1` is `TRUE`) and `block_write` is called with the `raw` flag set (`TRUE`), the block of bytes to be transferred will be transferred using DMA. Currently, DMA is only supported on the HP 9000 Models 825 and 835 SPUs with a A1047A interface card. The `gescape` will output an error if A1047A hardware is not present. If the application continues and calls `block_write` with the `raw` flag set, a warning will be generated and a standard `block_write` will be performed. There are many constraints on data alignment that are required by the A1047A hardware. It is the user's responsibility to properly align his data in a contiguous block of memory, lock the data in RAM, and flush the data cache. The following alignment restrictions exist on the parameters to the `block_write` call:

- `x_dest` must be on a 16-pixel boundary in the frame buffer (that is, its address must be modulo 16).
- `length_x` must be a multiple of 32.
- `pixel_data` must be on a 32-byte boundary in main memory (that is, its main memory address must be modulo 32).
- Do not specify parameters that would result in DMA outside the device coordinate range (0–1279 in X direction, 0–1023 in Y direction). This will result in unpredictable behavior.
- The use of device coordinates (`dcblock_write`) is recommended to make it easier to follow the alignment restrictions.

Note The data alignment restrictions are relative to the screen. When using an X window the position of the window relative to the origin of the device must be considered.

The following additional restrictions apply while doing `block_write` with DMA:

- Clipping operations are disabled when doing DMA to avoid sending unaligned data to the hardware. Setting the `raw` flag insures that clipping is not done, independent of whether or not Starbase clipping has been enabled.

- The replacement rule value of three is always used during DMA regardless of what was selected by **drawing_mode**. The value selected by **drawing_mode** remains unchanged and is used for all other non-DMA operations.

The **R_DMA_MODE gescape** is only supported for byte per pixel data and is mutually exclusive with the **R_BIT_MODE gescape**. If bit mode is turned on and an attempt is made to do DMA, an error will occur and DMA mode will not be set. If a call is made to turn on DMA mode and A1047A hardware is not present, an error will occur and DMA mode will be turned on anyway. When **block_write** is called with the **raw** flag set, a warning will be generated and a normal block write operation will be performed.

If **arg1** is **TRUE** (1), **raw** mode is DMA transfer. If **arg1** is **FALSE** (0), normal **raw** mode is used.

The **arg2** parameter is ignored.

The following presents a method for aligning one's data and performing a DMA transfer:

C Syntax

```
#include <starbase.c.h>
#include <sys/lock.h>
#define PATTERN_SIZE (320*100)

main(argc,argv)
int argc;
char *argv[];
{
 int fildes;
 gescape_arg arg1, arg2;  /* Gescape arguments */
 char *buf, *buf32;  /* Initial and aligned data pointers */

 fildes = gopen("/dev/crt",OUTDEV,"hp98731",INIT);

 arg1.i[0] = 1;
 gescape(fildes, R_DMA_MODE, &arg1, &arg2);  /* Enable DMA mode */

 /* Allocate 32-byte alligned buffer */
 allocate_aligned32(&buf, &buf32, PATTERN_SIZE);
```

```
/* DMA data must be locked in memory (see framebuf(7) and plock(2)). */
if (plock(DATLOCK)) {
   printf("*** Data lock failed.\n");
   /* If this fails, make sure you are executing with user id
      of root, or have group privileges via setprivgrp. */
   exit(1);
}

/* Load your data into buf32 here */

/* Flush the data cache before DMA (see section on data cache
         flushing in framebuf(7)) */

/* Write the buffer out to a different part of screen.
   Note that x_dest must be a multiple of 16 and length_x
   must be a multiple of 32. The raw flag is 1 to indicate
   use of DMA.  */
dcblock_write(fildes, 48, 0, 320, 100, buf32, 1);

gclose(fildes);
}

/* Allocate a 32-byte alligned buffer. */
/* This routine is also used by the Fortran and Pascal examples below. */
allocate_aligned32(initial,aligned,datasize)
char **initial, **aligned;
int datasize;
{
 *initial = (char *)malloc (datasize + 32);
 *aligned = (char *)(((int)*initial + 31) & 0xffffffe0);
}
```

FORTRAN77 Syntax

Since FORTRAN77 has no generalized pointer type, to use the aligned address you must make the address the base of an array. This is done by sending the address as a parameter to the `do_dma` subroutine that thinks the parameter is an array of the appropriate size. Then use the `"alias"` compiler directive to make the main program think that the parameter is being sent by value. The `"alias"` compiler directive must be inside the main program because if it is outside, the compiler realizes that you are trying to access a reference parameter as called by value.

```
 include '/usr/include/starbase.f1.h'

$alias allocate_aligned32 (%ref,%ref,%val)
$alias plock (%val)

 program main

$alias do_dma (%ref, %val)

 integer*4 fildes, error, arg1(64), arg2(64)
 integer*4 buf, buf32
 include '/usr/include/starbase.f2.h'
 integer*4 plock

 fildes = gopen('/dev/crt'//char(0),OUTDEV,
      +'hp98731'//char(0),INIT)

C Turn on DMA mode
 arg1(1) = 1
        call gescape(fildes, R_DMA_MODE, arg1, arg2)

C Allocate 32-byte alligned buffer
        call allocate_aligned32(buf, buf32, 32000)
```

```
C DMA data must be locked in memory (see framebuf(7) and plock(2)).
 if (plock(4) .ne. 0) then
    print *,"*** Data lock failed."
C  If this fails, make sure you are executing with user id
C  of root, or have group privileges via setprivgrp.
    stop
 endif

C Pass address of 32-byte aligned buffer to routine that does DMA
 call do_dma(fildes, buf32)

 error = gclose(fildes)

 END

 subroutine do_dma(fildes, buf32)

 integer*4 fildes
 character buf32(32000)

C Load your data into buf32 here

C  Flush the data cache before DMA (see section data cache
C       flushing in framebuf(7))

C  Write the buffer out to a different part of screen.
C  Note that x_dest must be a multiple of 16 and length_x
C  must be a multiple of 32. The raw flag is 1 to indicate
C  use of DMA.
 call dcblock_write(fildes, 48, 0, 320, 100, buf32, 1)


 END
```

Pascal Syntax

```
$standard_level 'hp_modcal'$
program main (output);
$include '/usr/include/starbase.p1.h'$

const
 PATTERN_SIZE = 320*100;
 DATLOCK = 4;

type
 datatype = packed array [0..(PATTERN_SIZE + 31)] of gbyte;
 data_ptr = ^datatype;

var
 fildes, error : integer;
 arg1, arg2 : gescape_arg; {Gescape arguments}
 buf, buf32 : data_ptr;  {Initial and aligned data pointers}

$include '/usr/include/starbase.p2.h'$

procedure allocate_aligned32 (var initial, aligned : data_ptr;
         datasize : integer); external;
function plock(op : integer) : integer;   external;

begin
 fildes := gopen('/dev/crt',OUTDEV,'hp98731',INIT);

 arg1.i[1] := 1;
 gescape(fildes, R_DMA_MODE, arg1, arg2); {Enable DMA mode}

        { Allocate 32-byte alligned buffer }
 allocate_aligned32(buf, buf32, PATTERN_SIZE);
```

```
        { DMA data must be locked in memory (see framebuf(7) and plock(2)). }
 if (plock(DATLOCK)<>0) then
 begin
    writeln('*** Data lock failed.');
    {  If this fails, make sure you are executing with user id
       of root, or have group privileges via setprivgrp. }
    halt(1);
 end;

 { Load your data into buf32 here }

 { Flush the data cache before DMA (see section on data cache
          flushing in framebuf(7)) }

 { Write the buffer out to a different part of screen.
   Note that x_dest must be a multiple of 16 and length_x
   must be a multiple of 32. The raw flag is 1 to indicate
   use of DMA. }
 dcblock_write(fildes, 48, 0, 320, 100, buf32^, 1);

 error := gclose(fildes);
end.
```

R_ECHO_CONTROL

The ⟨*op*⟩ parameter is R_ECHO_CONTROL.

The HP 98730 workstation provides both hardware and software cursors. Normally, right before cursors are used the first time, the driver tries to allocate the hardware cursor. If the hardware cursor is already being used by another process, the driver uses software cursors. This attempt to use the hardware cursor is only done once by the driver the first time it uses cursors. If the driver gets access to the hardware cursor, the hardware cursor is not relinquished until gclose time. If the driver did not get access to the hardware cursor, the driver will never try for the hardware cursor again, even if cursors are turned off and back on again. Instead it will use software cursors until gclose time.

When initializing cursor state, the driver will try to allocate the hardware cursor whenever any routine is called that modifies cursor state. These routines are:

- define_raster_echo, and echo_type.
- Any of the gescapes R_DEF_ECHO_TRANS, R_ECHO_MASK, R_ECHO_FG_BG_COLORS, and R_OV_ECHO_COLORS.

If this gescape is called before the first time cursors are used, it can be used to control whether hardware or software cursors will be used at cursor initialization time. If this gescape is called after cursors have been used, it can be used to determine what type of cursors the driver is using.

Input to this gescape is arg1[0] which contains a flag that can have one of the following three values:

- REQUEST_HW_ECHO (value of 1).

  If arg1[0] is REQUEST_HW_ECHO and cursors have already been initialized, the driver will attempt to allocate the hardware cursor for future usage. arg2[0] is returned 1 if the hardware cursor allocation was successful and hardware cursors will be used, otherwise it returns 0 if software cursors will be used.

  If arg1[0] is REQUEST_HW_ECHO and cursors have already been initialized, arg2[0] will contain 1 if hardware cursors are being used. Otherwise, arg2[0] will return 0 if software cursors are being used, and the driver will not attempt to allocate the hardware cursor.

- REQUEST_SW_ECHO (value of 2).

  If arg1[0] is REQUEST_sw_ECHO and cursors have not yet been initialized, the driver will use software cursors and not attempt to allocate the hardware cursor. If software cursors are being used, arg2[0] will be returned 0.

  If arg1[0] is REQUEST_SW_ECHO and cursors have already been initialized, arg2[0] will contain 1 if hardware cursors are being used, otherwise arg2[0] will return 0 if software cursors are being used, and the driver will not attempt to relinquish the hardware cursor if it was being used.

- FORCE_HW_ECHO (value of 3).

  If arg1[0] is FORCE_HW_ECHO and cursors have not yet been initialized, the driver will attempt to allocate the hardware cursor for future use. If the hardware cursor allocation was successful, arg2[0] is returned and 0 if not. Even if the driver could not successfully allocate the hardware cursor, it will use the hardware cursor.

  If arg1[0] is FORCE_HW_ECHO and cursors have already been initialized, arg2[0] will contain 1 if hardware cursors are being used, otherwise arg2[0] will return 0 if software cursors are being used, and the driver will not attempt to allocate or use the hardware cursor.

Note The FORCE_HW_ECHO is a dangerous mode and should only be used when the user knows that other processes will not be attempting to update the hardware cursor simultaneously. Refer to the driver section on cursor usage for a more complete discussion of using this mode.

C Syntax

```
/* gescape_arg is typedef defined in starbase.c.h */

gescape_arg arg1, arg2;
   :
arg1.i[0]=REQUEST_HW_ECHO;
gescape(fildes,R_ECHO_CONTROL,&arg1,&arg2);
```

FORTRAN77 Syntax

```
integer*4 arg1(64),arg2(64)
arg1(1)=REQUEST_HW_ECHO
call gescape(fildes,R_ECHO_CONTROL,arg1,arg2)
```

Pascal Syntax

```
{gescape_arg is defined in starbase.p1.h}

var
   arg1,arg2:gescape_arg;
      :
begin
   arg1.i[1] := REQUEST_HW_ECHO
   gescape(fildes,R_ECHO_CONTROL,arg1,arg2);
```

R_ECHO_FG_BG_COLORS

The ⟨*op*⟩ parameter is R_ECHO_FG_BG_COLORS

The HP 98730 workstation and HP 98731 accelerator provide both hardware and software cursors. Hardware cursors give the best performance because cursors do not have to be "picked up" and "put down" around every graphics output operation, since hardware cursors have their own dedicated graphics planes. Software cursors are cursors that are written to the same frame buffer area that graphics are also currently using. Therefore, they have to be "picked up" and "put down" around graphics output operations. This gives lower performance.

This gescape lets the user define color attributes for cursors. The functionality of this gescape depends on the mode of cursors that is currectly active. The three modes are:

- Hardware cursors
- Non-overlayed software cursors are written in the same graphics planes that graphics is currently being written to.
- Overlayed software cursors are written to the fourth overlay plane. This mode is only supported if opened to the image planes.

Refer to the gescape R_OVERLAY_ECHO for more information on the location of software cursors.

When initializing cursor state, the driver will try to allocate the hardware cursor whenever any routine is called that modifies the cursor state. These routines are:

- define_raster_echo, and echo_type.
- Any of the gescapes R_DEF_ECHO_TRANS, R_ECHO_MASK, R_ECHO_FG_BG_COLORS, and R_OV_ECHO_COLORS.

For explicit control of allocations of the hardware cursor, refer to the gescape R_ECHO_CONTROL.

Further discussion of this gescape is categorized by the mode of cursor being used. Such as: Hardware Cursor, Overlayed Software Cursor, or Non Overlayed Software Cursor.

Hardware Cursors

If using hardware cursors, then vector cursors only have a foreground color, and raster cursors have both a foreground color and a background color.

There are two color maps for the hardware cursors which alternate every 133ms. Therefore, for each of the foreground colors and background colors, two colors can be specified for each and the cursor will blink between the two specified colors.

As input to this `gescape`, there is a flag associated with the foreground color and a flag associated with the background color. If this flag has the value 0, it means "do not modify the color". If this flag has the value 1, it means "modify the color". If the flag has the value 2, the foreground (or background) of the raster cursor should be treated as transparent.

For example: If the flag value for the foreground color is defined as transparent, for every zero value in the raster cursor pattern, the graphics image behind the cursor will be visible. Even if the foreground color is being defined as transparent, red, green, and blue values should be provided because the foreground colors will be used when switching back to vector cursors.

The initial state of hardware cursors is a white foreground color, and a transparent background for raster cursors. If this `gescape` is used to redefine the state of the foreground or background transparency for hardware raster cursors, `define_raster_echo` must be called to ensure proper initialization of the hardware raster cursor bitmaps.

One final piece of information is needed for hardware cursors. This is an index value to associate with the raster cursor background color. This index is used when a raster echo pattern is being defined to the hardware cursor (see `define_raster_echo`). During this definition, every value in the raster definition that has the background color index value specified by this `gescape` will be defined to the hardware as the background color pixel. Every other value found in the raster pattern will be defined as a foreground pixel for the raster cursor. The default index value defined at `gopen` time is 0.

All the data for this `gescape` is provided in `arg1` and is all in floating point notation. The order of the data is:

`arg1.f[0]` Flag for foreground color.
0.0 = Do not alter current foreground color.
1.0 = Alter current foreground color.

2.0 = Foreground is transparent. Even if foreground is transparent, the following red, green, blue values are defined to the hardware cursor to be used with vector cursors.

`arg1.f[1]` Primary color map red value.

`arg1.f[2]` Primary color map green value.

`arg1.f[3]` Primary color map blue value.

`arg1.f[4]` Unused.

`arg1.f[5]` Secondary color map red value.

`arg1.f[6]` Secondary color map green value.

`arg1.f[7]` Secondary color map blue value.

`arg1.f[8]` Unused.

`arg1.f[9]` Flag for background color.
0.0 = Do not alter current background color.
1.0 = Alter current background color.
2.0 = Background is transparent.

`arg1.f[10]` Primary color map red value.

`arg1.f[11]` Primary color map green value.

`arg1.f[12]` Primary color map blue value.

`arg1.f[13]` Secondary color map red value.

`arg1.f[14]` Secondary color map green value.

`arg1.f[15]` Secondary color map blue value.

`arg1.f[16]` Index to use for background pixels in raster pattern definition.

Overlayed Software Cursors

Overlayed software cursors are cursors in the fourth overlay plane. Refer to the `gescape R_OVERLAY_ECHO` for more discussion on overlayed cursors. Overlayed software cursors do not need to be "picked up" and "put down" again around graphics output, since they are not in the same graphics planes currently being used by graphics. Threfore, they offer better performance than non overlayed software cursors.

For overlayed raster cursors a foreground color and a cursor mask can be defined. For defining cursor masks refer to the R_ECHO_MASK or the R_DEF_ECHO_TRANS gescapes.

For overlayed software cursors there are two color maps which alternate every 133ms. Thus, for the cursor color, two colors can be defined and the cursor will blink between the two colors.

For foreground color definition, a transparency value is also given. If the transparency value is 1.0, the pixel color is forced to the color specified by the red, green, and blue values provided by the foreground color. If the transparency value is 0.0, the pixel color will be the color in the graphics planes "behind" the overlay planes.

Calling this gescape causes the driver to update the overlay color map so that for all entries that are transparent, the cursor will be seen. Therefore, even though the cursor is written to the fourth overlay plane, it appears to be in the image plane behind the overlay planes. If another process opened to the overlay planes defines another transparent entry using the R_TRANSPARENCY_INDEX gescape, calling this gescape will cause the color map to be updated so that the cursor will also be seen in this new region of transparency. If this gescape is not called after defining a new transparency entry in the overlay planes, the cursor for the image planes will not be seen in regions of the new transparency index.

One final piece of information is needed for overlayed software cursors. This is an index value to associate with the raster cursor background color. This index is used when a raster echo pattern is being defined (see define_raster_echo). During this definition, every value in the raster definition that has the background color index value specified by this gescape will be defined to the driver as the background color pixel. Every other value found in the raster pattern will be defined as a foreground pixel for the raster cursor. The default index value defined at gopen time is 0.

All the data for this gescape is provided in arg1 and is all in floating point notation. The order of the data is:

arg1.f[0]	Flag for foreground color. 0.0 = Do not alter current foreground color. 1.0 = Alter current foreground color. 2.0 = Unused.
arg1.f[1]	Primary color map red value.

`arg1.f[2]` Primary color map green value.

`arg1.f[3]` Primary color map blue value.

`arg1.f[4]` Transparency bit. 0.0 = Transparent. Force pixel to color in image planes behind the overlay planes.
1.0 = Not transparent. Force pixel to red, green, blue color.

`arg1.f[5]` Secondary color map red value.

`arg1.f[6]` Secondary color map green value.

`arg1.f[7]` Secondary color map blue value.

`arg1.f[8]` Transparency bit.
0.0 = Transparent. Force pixel to color in image planes behind the overlay planes.
1.0 = Not transparent. Force pixel to red, green, blue color.

`arg1.f[9]` Unused.

`arg1.f[10]` Unused.

`arg1.f[11]` Unused.

`arg1.f[12]` Unused.

`arg1.f[13]` Unused.

`arg1.f[14]` Unused.

`arg1.f[15]` Unused.

`arg1.f[16]` Index to use for background pixels in raster pattern definition.

Non-Overlayed Software Cursors

Non-overlayed software cursors are cursor written to the same planes that graphics are currently being written to. These cursors need to be "picked up" before graphics output, and "put down" again after graphics output, thus, they are slower. Non-overlayed software cursors are not available on the HP 98731 device. In order to maximize performance all cursors are overlayed.

This `gescape` is not supported for non-overlayed software cursors because cursor colors can not be defined. When writing these software cursors to the frame buffer, a replacement rule of not-destination is used for vector cursors. For raster cursors, the raster bitmap for the cursor is written to the graphics planes. Thus,

the raster cursor color depends on the values in the raster cursor bitmap and the current color table definition.

However, for non-overlayed software cursors, a raster echo mask can be defined. Refer to the gescape R_DEF_ECHO_TRANS or R_ECHO_MASK for more discussion of raster echo masks.

Examples and Syntax

Following are two examples. The first example defines a foreground color blinking between red and green and a background color of blue. It assumes that access to the hardware cursor has been granted.

The second example defines a transparent foreground and a background color of red. It also assumes access to the hardware cursor has been granted.

C Syntax

Example 1:

```
/* gescape_arg is typedef defined in starbase.c.h */

gescape_arg arg1, arg2;
   :
arg1.f[0]=1.0; /* Set flag indicating define foreground color */
arg1.f[1]=1.0; /* Red value for primary color map */
arg1.f[2]=0.0; /* Green value for primary color map */
arg1.f[3]=0.0; /* Blue value for primary color map */
arg1.f[4]=0.0; /* Transparency flag.  Ignored since we are using hardware */
               /* cursors. */
arg1.f[5]=0.0; /* Red value for secondary color map */
arg1.f[6]=1.0; /* Green value for secondary color map */
arg1.f[7]=0.0; /* Blue value for secondary color map */
arg1.f[8]=0.0; /* Transparency bit.  Ignored since we are using hardware */
               /* cursors. */
arg1.f[9]=1.0; /* Set flag indicating define background color */
arg1.f[10]=0.0; /* Red value for primary color map */
arg1.f[11]=0.0; /* Green value for primary color map */
arg1.f[12]=1.0; /* Blue value for primary color map */
arg1.f[13]=0.0; /* Red value for secondary color map */
arg1.f[14]=0.0; /* Green value for secondary color map */
arg1.f[15]=1.0; /* Blue value for secondary color map */
arg1.f[16]=0.0; /* Cursor background color index */
gescape(fildes,R_ECHO_FG_BG_COLORS,&arg1,&arg2);
```

```
/* A call to define_raster_echo should follow this since it changed the */
/* background from the default configuration of transparent to a defined */
/* color. */
```

Example 2:

```
/* gescape_arg is typedef defined in starbase.c.h */

gescape_arg arg1, arg2;
   :
arg1.f[0]=2.0; /* Set flag indicating transparent foreground */
/* The following rgb values will be used for vector cursors */
arg1.f[1]=1.0; /* Red value for primary color map */
arg1.f[2]=0.0; /* Green value for primary color map */
arg1.f[3]=0.0; /* Blue value for primary color map */
arg1.f[4]=0.0; /* Transparency flag.  Ignored since we are using hardware */
               /* cursors. */
/* The following rgb values will be used for vector cursors */
arg1.f[5]=1.0; /* Red value for secondary color map */
arg1.f[6]=0.0; /* Green value for secondary color map */
arg1.f[7]=0.0; /* Blue value for secondary color map */
arg1.f[8]=0.0; /* Transparency bit.  Ignored since we are using hardware */
               /* cursors. */
arg1.f[9]=1.0; /* Set flag indicating define background color */
arg1.f[10]=1.0; /* Red value for primary color map */
arg1.f[11]=0.0; /* Green value for primary color map */
arg1.f[12]=0.0; /* Blue value for primary color map */
arg1.f[13]=1.0; /* Red value for secondary color map */
arg1.f[14]=0.0; /* Green value for secondary color map */
arg1.f[15]=0.0; /* Blue value for secondary color map */
arg1.f[16]=0.0; /* Cursor background color index */
gescape(fildes,R_ECHO_FG_BG_COLORS,&arg1,&arg2);
/* A call to define_raster_echo should follow this since it changed the */
/* foreground to be transparent. */
```

FORTRAN77 Syntax

Example 1:

```
      real arg1(64),arg2(64)
   arg1(1)=1.0
   arg1(2)=1.0
   arg1(3)=0.0
   arg1(4)=0.0
   arg1(5)=0.0
   arg1(6)=0.0
   arg1(7)=1.0
   arg1(8)=0.0
   arg1(9)=0.0
   arg1(10)=1.0
   arg1(11)=0.0
   arg1(12)=0.0
   arg1(13)=1.0
   arg1(14)=0.0
   arg1(15)=0.0
   arg1(16)=1.0
   arg1(17)=0.0
   call gescape(fildes,R_ECHO_FG_BG_COLORS,arg1,arg2)
C  A call to define_raster_echo should follow this since it changed the
C  background from the default configuration of transparent to a defined
C  color.
```

Example 2:

```
      real arg1(64),arg2(64)
   arg1(0)=2.0
C  the following rgb values will be used for vector cursors
   arg1(1)=1.0
   arg1(2)=0.0
   arg1(3)=0.0
   arg1(4)=0.0

C  the following rgb values will be used for vector cursors
   arg1(5)=1.0
   arg1(6)=0.0
   arg1(7)=0.0
   arg1(8)=0.0
   arg1(9)=1.0
   arg1(10)=1.0
   arg1(11)=0.0
   arg1(12)=0.0
```

```
      arg1(13)=1.0
      arg1(14)=0.0
      arg1(15)=0.0
      arg1(16)=0.0
      call gescape(fildes,R_ECHO_FG_BG_COLORS,arg1,arg2)
C     A call to define_raster_echo should follow this since it changed the
C     foreground to be transparent.
```

Pascal Syntax

Example 1:

```
{ gescape_arg is defined in starbase.p1.h }

var arg1, arg2: gescape_arg;
   :
arg1.f[0]:=1.0; { Set flag indicating define foreground color }
arg1.f[1]:=1.0; { Red value for primary color map }
arg1.f[2]:=0.0; { Green value for primary color map }
arg1.f[3]:=0.0; { Blue value for primary color map }
arg1.f[4]:=0.0; { Transparency flag.  Ignored since we are using hardware }
                { cursors. }
arg1.f[5]:=0.0; { Red value for secondary color map }
arg1.f[6]:=1.0; { Green value for secondary color map }
arg1.f[7]:=0.0; { Blue value for secondary color map }
arg1.f[8]:=0.0; { Transparency bit.  Ignored since we are using hardware }
                { cursors. }
arg1.f[9]:=1.0; { Set flag indicating define background color }
arg1.f[10]:=0.0; { Red value for primary color map }
arg1.f[11]:=0.0; { Green value for primary color map }
arg1.f[12]:=1.0; { Blue value for primary color map }
arg1.f[13]:=0.0; { Red value for secondary color map }
arg1.f[14]:=0.0; { Green value for secondary color map }
arg1.f[15]:=1.0; { Blue value for secondary color map }
arg1.f[16]:=0.0; { Define 0 as background index }
gescape(fildes,R_ECHO_FG_BG_COLORS,arg1,arg2);
{ A call to define_raster_echo should follow this since it changed the }
{ background from the default configuration of transparent to a defined }
{ color. }
```

Example 2:

```
{ gescape_arg is defined in starbase.p1.h }

var arg1, arg2: gescape_arg;
   :
arg1.f[0]:=2.0; { Set flag indicating transparent foreground }
{the following rgb values will be used for vector cursors }
arg1.f[1]:=1.0; { Red value for primary color map }
arg1.f[2]:=0.0; { Green value for primary color map }
arg1.f[3]:=0.0; { Blue value for primary color map }
arg1.f[4]:=0.0; { Transparency flag.  Ignored since we are using hardware }
              { cursors. }
{ The following rgb values will be used for vector cursors }
arg1.f[5]:=1.0; { Red value for secondary color map }
arg1.f[6]:=0.0; { Green value for secondary color map }
arg1.f[7]:=0.0; { Blue value for secondary color map }
arg1.f[8]:=0.0; { Transparency bit.  Ignored since we are using hardware }
              { cursors. }
arg1.f[9]:=1.0; { Set flag indicating define background color }
arg1.f[10]:=1.0; { Red value for primary color map }
arg1.f[11]:=0.0; { Green value for primary color map }
arg1.f[12]:=0.0; { Blue value for primary color map }
arg1.f[13]:=1.0; { Red value for secondary color map }
arg1.f[14]:=0.0; { Green value for secondary color map }
arg1.f[15]:=0.0; { Blue value for secondary color map }
arg1.f[16]:=0.0; { Define 0 as background index }
gescape(fildes,R_ECHO_FG_BG_COLORS,arg1,arg2);
{ A call to define_raster_echo should follow this since it changed the }
{ foreground to be transparent }
```

R_ECHO_MASK

The ⟨*op*⟩ parameter is R_ECHO_MASK

This gescape allows the user to define a mask for raster cursors. It provides the same functionality as R_DEF_ECHO_TRANS, except that the input data is byte aligned on row boundaries. It is suggested that R_ECHO_MASK be used instead of R_DEF_ECHO_TRANS for slightly better performance. An echo mask is used to determine which bits of the raster cursor pattern are visible over the graphics background. The mask is assumed to have the same height and width as the current raster cursor. The mask is arranged in a packed array as one-bit per pixel. Each byte represents eight pixels, and row boundaries are byte aligned. If the bit in the mask is set, the corresponding pixel location in the raster cursor will be visible. If the mask bit is zero, the corresponding pixel of the raster cursor will not be applied to the frame buffer.

After this gescape has been called, the echo mask will be used to draw the current raster cursor until another raster cursor is defined with a call to define_raster_echo. If define_raster_echo is called, it is necessary to follow that call with another call to this gescape to use an echo mask.

If defining a mask to be used with the hardware cursor on a HP 98730 worstation, this gescape should be used for better performance.

With the hp98730 device driver, echo masks cannot be used in a graphics window if the echo currently being used is not the hardware cursor, or the echo is not overlayed in the fourth overlay plane.

The arg1 parameter points to the echo mask.

The arg2 parameter is ignored.

The default raster cursor is a 8×8 pattern. The following example defines a echo mask for the default raster cursor that is 108 in size. The default raster cursor and echo mask are justified in the upper left 8×8 square. An extra two pixels on the right hand side are being included to demonstrate how the data is byte aligned on row boundaries.

C Syntax

```
/* gescape_arg is typedef defined in starbase.c.h */

gescape_arg arg1, arg2;
   :
arg1.i[0] = 0xF00C00A0;
arg1.i[1] = 0x09000800;
arg1.i[2] = 0x40020010;
gescape(fildes,R_ECHO_MASK,&arg1,&arg2);
```

FORTRAN77 Syntax

```
interger*4 arg1(64),arg2(64)
arg1(1)= Z'F00C00A0'
arg1(2)= Z'09000800'
arg1(3)= Z'40020010'
call gescape(fildes,R_ECHO_MASK,arg1,arg2)
```

Pascal Syntax

```
 {gescape_arg is defined in starbase.p1.h}

type
  mask_def = array [1..2] of integer;
  mask_ptr = ^mask_def;
  ptrunion = record case integer of
 1 :(a : mask_ptr);
 2 :(b : gescape_arg)
      end;

var
  arg1,arg2,null:gescape_arg;
  pointers:ptrunion;
  mask : mask_def;

 begin
 mask[1] := hex('F00C00A0');
 mask[2] := hex('09000800');
 mask[3] := hex('40020010');
 pointers.a := ^mask;
    gescape(fildes,R_ECHO_MASK,pointers.b,null);
```

R_FULL_FRAME_BUFFER

The ⟨*op*⟩ parameter is `R_FULL_FRAME_BUFFER`.

This `gescape` allows access to the off screen area of the frame buffer after the `set_p1_p2` procedure is called.

The `arg1` parameter is a flag, when `TRUE`(1), the physical limits of the device are set to maximum frame buffer memory size. When `FALSE`(0), the physical limits are set to the visual screen area.

The `arg2` parameter is ignored.

Note

Care should be taken when using this `gescape` since other processes can access the frame buffer and the driver may use some off-screen memory. HP Windows/9000 uses offscreen for its fonts and sprite (HP Windows/9000 's tracking echo). X11 also uses offscreen for its fonts and sprite, as well as pixmaps and backing store (retained rasters). Notice: HP Windows/9000 and X11 leave the user with very little extra offscreen memory to use. Refer to the "Device Description" segment in the device drivers section for details of frame buffer sizes and current usage of offscreen memory by Starbase.

The specification for use of this area by Starbase, HP Windows/ 9000 and X11 may change for future releases. As a result, more offscreen memory may be required than is currently used.

Hewlett-Packard does not guarantee that the use of offscreen frame buffer memory will remain the same for future releases of Starbase, HP Windows/9000, and X11.

The following sections describe the Windows/9000 usage of offscreen frame buffer memory on different devices.

HP 300h Device

Windows/9000 uses the first 32 lines of off-screen frame buffer memory for the sprite. Only the first 96 bytes of each of these lines is used; the rest is unused.

The remaining lines, up to the last 16 used by Starbase, are used for window system fonts.

HP 300l Device

Windows/9000 uses the first 32 lines of off-screen frame buffer memory for the sprite. Only the first 192 bytes of each of these lines is used; the rest is unused. The remaining lines, up to the last 16 used by Starbase, are used for window system fonts.

HP 98700 Device

Windows/9000 uses the last 64 lines of off-screen frame buffer memory for the sprite. The 64×64 area immediately below this sprite area is unused. The fonts are contained in the first 960 bytes of all 256 lines of off-screen memory.

HP 98550/HP 98556 Device

Windows/9000 uses the first 96 bytes of the first 32 lines of off-screen frame buffer memory for the sprite. The next 656 bytes of each of these lines are always used for dithered-fill patterns. The rest of each line is unused. All remaining lines are used by the window system's fonts. Starbase raster echoes consume 64 lines by 128 bytes per gopen but are allocated only as needed.

HP 98720 Device

Windows/9000 uses the first 96 bytes of the first 32 lines of off-screen overlay planes frame buffer memory for the sprite. The next 656 bytes of each of these lines is always used for dithered-fill patterns. The rest of each line is unused. All remaining lines are used by the window system's fonts.

HP 98730/HP 98731 Device

A global resource manager is used to manage allocation and deallocation of overlay planes offscreen memory for fonts, windex system sprites, and dithered-fill patterns. Refer to the device driver section for a description of offscreen memory usage.

C Syntax

```
/* gescape_arg is typedef defined in starbase.c.h */

gescape_arg arg1, arg2;
   :
arg1.i[0] = 1;
gescape(fildes,R_FULL_FRAME_BUFFER,&arg1,&arg2);
set_p1_p2(fildes,FRACTIONAL,0.0,0.0,0.0,1.0,1.0,1.0);
```

FORTRAN77 Syntax

```
integer*4 arg1(1)=1
call gescape(fildes,R_FULL_FRAME_BUFFER,arg1,arg2)
call set_p1_p2(fildes,FRACTIONAL,0.0,0.0,0.0,1.0,1.0,1.0);
```

Pascal Syntax

```
{gescape_arg is defined in starbase.p1.h}

var
   arg1,arg2 : gescape_arg;
   :
begin
   arg1.i[1] = 1;
   gescape(fildes,R_FULL_FRAME_BUFFER,arg1,arg2);
   set_p1_p2(fildes,FRACTIONAL,0.0,0.0,0.0,1.0,1.0,1.0);
```

R_GET_FRAME_BUFFER

The ⟨*op*⟩ parameter is R_GET_FRAME_BUFFER.

This gescape will read the address of the device's frame buffer and control space.

The arg1 parameter is ignored.

The arg2[0] parameter will return the address of the device's control space. The arg2[1] parameter will return the address of the upper-left corner of the device's frame buffer.

Note Be careful when using this gescape since other processes can also access the frame buffer. You *must* call the R_LOCK_DEVICE gescape before attempting to access the frame buffer in this way. The R_UNLOCK_DEVICE gescape should be called when finished accessing the frame buffer.

In order to avoid any conflict with the current graphics process, a MAKE_PICTURE_CURRENT call *must* be done before accessing the hardware directly. This will ensure all buffers are flushed.

See the "Device Description" segment of the appropriate driver section for details on frame buffer organization.

The following examples draw a line in the frame buffer using this gescape. The bytes-per-row multiplier (2048 in the following example) is device-dependent. See the appropriate driver section for the correct width of the frame buffer memory.

Series 800 Dependency

These examples are written for the Series 300 computers. On the Series 800 computers, the frame buffer arrays would need to be changed to arrays of integers since the IO is 32 bits wide. In the C syntax example, the line `register unsigned char *frame` would become `register unsigned int *frame`.

C Syntax

```
#include <starbase.c.h>

main(argc,argv)
int argc;
```

```
char **argv;
{
      register int fildes,i;
      register unsigned char *frame;
      gescape_arg arg1, arg2;

      /* Open device using Path and driver name passed on command line */
      fildes = gopen(argv[1],OUTDEV,argv[2],INIT);

      /* Get address of frame buffer */
      gescape(fildes,R_GET_FRAME_BUFFER,&arg1,&arg2);
      frame = (unsigned char *) arg2.i[1];

      /* Lock the device before accessing frame buffer */
      gescape(fildes, R_LOCK_DEVICE,&arg1,&arg2);

      /* Draw a vertical line from (x=99,y=5) to (x=99,y=299) */
      for (i=5;i<300;i++)
            frame[99 + i*2048] = (char) 3;

      /* Unlock device */
      gescape(fildes,R_UNLOCK_DEVICE,&arg1,&arg2);

      /* Close the device and exit */
      gclose(fildes);
}
```

FORTRAN77 Syntax

Since FORTRAN77 has no generalized pointer type, to use the address you must make the address the base address of an array. This is done by sending the address as a parameter to a subroutine that thinks the parameter is an array of the appropriate size. Use the "alias" compiler directive to make the main program think that the parameter is being sent by value. The "alias" compiler directive must be inside the main program because if it is outside, the compiler realizes that you are trying to access a reference parameter as called by value.

```
        include '/usr/include/starbase.f1.h'
        program main
$alias doline (%val)

        integer*4 fildes,error,arg1(10),arg2(10)
        include '/usr/include/starbase.f2.h'

C       Open device, for driver of interest
        fildes = gopen('/dev/crt',OUTDEV,Driver_name,INIT)

C       Get frame buffer address
        call gescape(fildes,R_GET_FRAME_BUFFER,arg1,arg2)

C       Lock the device before accessing frame buffer
        call gescape(fildes,R_LOCK_DEVICE,arg1,arg2)

C       Pass address to routine which draws a line
        call doline(arg2(2))

C       Unlock device
        call gescape(fildes,R_UNLOCK_DEVICE,arg1,arg2)

C       Close device and exit
        error = gclose(fildes)
        END
        subroutine doline(frame)
```

```
      integer*2 frame(1024*2048/2)
      integer*4 i

C     draw line from (x=99,y=5) to (x=99,y=299)
      do 10 i = 5,299,1
 10   frame((99+1)/2 + i * 2048/2) = 3

      END
```

Pascal Syntax

```
program main(output);
$include '/usr/include/starbase.p1.h'$

type
  frame_buffer = array [0..maxint] of char;
  fb_ptr = ^frame_buffer;
  ptrunion = record case integer of
               1 :(a : array [1..2] of fb_ptr);
               2 :(b : gescape_arg)
             end;

var
  null:gescape_arg;
  pointers:ptrunion;
  frame : fb_ptr;
  fildes,i:integer;

$include '/usr/include/starbase.p2.h'$

begin
  { Open device from name in driver }
  fildes := gopen('/dev/crt',OUTDEV,driver,INIT);

  { Get frame buffer address }
  gescape(fildes,R_GET_FRAME_BUFFER,null,pointers.b);
  frame := pointers.a[2];
```

```
  { Lock device before accessing frame buffer}
  gescape(fildes,R_LOCK_DEVICE,null,null);

  { Draw line from (x=99,y=5) to (x=99,y=299) }
  for i := 5 to 299 do
    frame^[99 + i*2048] := chr(3);

  { Unlock device }
  gescape(fildes,R_UNLOCK_DEVICE,null,null);

  { Close and exit }
  error := gclose(fildes);
end.
```

R_GET_WINDOW_INFO

The ⟨*op*⟩ parameter is `R_GET_WINDOW_INFO`.

The window address of a Windows/9000 window can be used as the origin when accessing the frame buffer directly. Remember that each line of the frame buffer still has the same width.

The `arg1` parameter is ignored.

If the device is a window device, this procedure returns the address of the upper-left corner of the window in `arg2.i[0]`. If the window is currently unobscured (completely visible on the display), `arg2.i[1]` returns 1, otherwise, it returns 0.

If the device is the raw device, `arg2.i[0]` returns the frame buffer address, and `arg2.i[1]` returns 1 (not obscured).

C Syntax

```
/* gescape_arg is typedef defined in starbase.c.h */

gescape_arg arg1, arg2;
   ⋮
gescape(fildes,R_GET_WINDOW_INFO,&arg1,&arg2);
```

FORTRAN77 Syntax

```
integer*4 arg1(64),arg2(63)
      call gescape(fildes,R_GET_WINDOW_INFO,arg1,arg2)
```

Pascal Syntax

```
{ gescape_arg is defined in starbase.p1.h }
var
  arg1,arg2 : gescape_arg;

begin
  gescape(fildes,R_GET_WINDOW_INFO,arg1,arg2);
```

The following example program shows R_GET_WINDOW_INFO in action.

C Program Example (not robust - no error checking)

```
#include <starbase.c.h>

main()
{
    int fildes;
    gescape_arg arg1, arg2;
    char *win_addr;

    /* Open a window on an HP 300h device for output and initialize it */
    fildes = gopen("/dev/screen/window1",OUTDEV,"hp98720w",INIT);

    /* Lock down the device and keep it if it's unobscured */
    /* Otherwise, unlock it and sleep awhile until can check it again */
    arg1.i[0] = 1; /* Remove sprite if it's on */
    while (TRUE) {
        gescape(fildes,R_LOCK_DEVICE,&arg1,&arg2);
        gescape(fildes,R_GET_WINDOW_INFO,&arg1,&arg2);
        if (arg2.i[1] != 0) break;
        gescape(fildes,R_UNLOCK_DEVICE,&arg1,&arg2);
             /*  To insure that window becomes unobscured, you may want to do
   a "wtop(fildes,1)" routine, which must be preceeded by the
   "winit(fildes)" routine.  See the Programmer's Manual for the
   Window Library.                                          */
        sleep(5); /* wait 5 seconds before next check */
    }
    /* Now have device locked and know window's address in arg2.i[0] */

    win_addr = (char *)arg2.i[0];

    /*  For additional speed improvement.  Window must be unobscured or
              screen will get garbaged !!                        */
```

```
    arg1.i[0] = 0;
    gescape(fildes,SWITCH_SEMAPHORE,&arg1,&arg2);
    .
    .
    .
    /* Do anything desired to window; use inquire_sizes if need to know
       size of window in pixels */
    .
    .
    .
    arg1.i[0] = 1;
    gescape(fildes,SWITCH_SEMAPHORE,&arg1,&arg2);

    /* Unlock the device now that finished */
    gescape(fildes,R_UNLOCK_DEVICE,&arg1,&arg2);

    gclose(fildes);
}  /* All done */
```

R_LINE_TYPE

The ⟨*op*⟩ parameter is `R_LINE_TYPE`.

This `gescape` is supported on the following device drivers: `hp98720`, `hp98721`, `hp98730`, `hp98731`, and the Starbase Memory Driver.

This `gescape` allows you to specifically define the current line style and repeat length to be used for all subsequent line primitives. A 16-bit repeat pattern is accepted, as well as the repeat length to be used. This gescape will override the line style set by `line_type` *and* the repeat length set by `line_repeat_length`. Further calls to `line_type` *or* `line_repeat_length` will override both values set with this `gescape`. See `line_type` and `line_repeat_length` in your *Starbase Reference* manual.

Both parameters for this `gescape` are passed in `arg1`. The first (`arg1.i[0]`) is a 16-bit pattern which defines the repeating pattern for lines. Note that even though a 32-bit integer is passed to the subroutine, only the least significant 16 bits will be used. The second parameter (`arg1.i[1]`) is an integer value which is the repeat length of the line type pattern. The repeat length specifies how the pattern is scaled. If the repeat length is one, the 16-bit pattern will be used for the first 16 pixels of the next line that is drawn, and will then begin repeating for subsequent pixels. If the repeat length is two, the first bit in the repeat pattern will be used for the first two pixels in the next line, and so on. The effect is to stretch the pattern. For example, if the pattern were specified as `0xAAAA` (hexadecimal), and the repeat length were 1, lines would be drawn with alternating pixels on and off (a very fine dotted line.) If the repeat length was 2, and the same pattern were used, lines would be drawn with two pixels on, followed by two pixels off (a more coarse dotted line.)

C Syntax

```
/* gescape_arg is typedef defined in starbase.c.h */

gescape_arg arg1, arg2;

arg1.i[0]=0x1010;
arg1.i[1]=1;

gescape(fildes,R_LINE_TYPE,&arg1,&arg2);
```

FORTRAN77 Syntax

```
integer*4 arg1(4),arg2(1)
arg1(1)=Z'1010'
arg1(2)=1
call gescape(fildes,R_LINE_TYPE,arg1,arg2)
```

Pascal Syntax

```
{gescape_arg is defined in starbase.p1.h}

var
   arg1,arg2:gescape_arg;

begin
   arg1.i[1] := hex('1010');
   arg1.i[2] := 1;
   gescape(fildes,R_LINE_TYPE,arg1,arg2);
```

R_LOCK_DEVICE

The ⟨*op*⟩ parameter is `R_LOCK_DEVICE`.

This procedure locks the device associated with the specified file descriptor (`fildes`).

This `gescape` is useful when semaphores are to be turned off or the frame buffer is to be accessed directly using `R_GET_FRAME_BUFFER`, and the program needs exclusive use of the display. Once the device is locked, any program that uses the semaphore can not access the device until it is unlocked.

Both the `arg1` and `arg2` parameters are ignored.

The following warnings apply to the time between an `R_LOCK_DEVICE gescape` and an `R_UNLOCK_DEVICE gescape`.

- If the device is the Console ITE also, any output to the console (ie. printf to `/dev/console`) should not be done.
- A fork should not be done because child processes get confused as to whether they own the lock or not.
- If a lock is in effect, characters typed on the Console ITE may block the ITE and prevent the break key from interrupting until the lock is released.
- The application should not perform any Starbase input (polling, `track_on`, or `track_off`) from a window device or to an output device on the same display as the lock.
- The application should not invoke the Windows/9000 `wgetlocator` procedure.
- The application should not use any X window system calls that access the X server.
- Signals with signal handlers installed may be masked until the lock is released. Changing the signal mask may or may not affect this masking by the graphics system. The signal mask may be changed again by unlocking the device. Do not change the signal mask yourself while the device is locked.

C Syntax

```
/* gescape_arg is typedef defined in starbase.c.h */

gescape_arg arg1, arg2;

gescape(fildes,R_LOCK_DEVICE,&arg1,&arg2);
```

FORTRAN77 Syntax

```
integer*4 arg1(64),arg2(64)
call gescape(fildes,R_LOCK_DEVICE,arg1,arg2)
```

Pascal Syntax

```
{gescape_arg is defined in starbase.p1.h}

var
arg1, arg2 : gescape_arg;

begin
gescape(fildes,R_LOCK_DEVICE,arg1,arg2);
```

C Example Program

The following program locks the device before disabling semaphores. Locking the device guarantees that this process has sole access to the device (assuming all other programs have semaphores turned on - the default). Disabling semaphores makes the program run slightly faster because an attempt to lock the device is no longer done before each output (or output buffer) to the device.

```
#include <starbase.c.h>

main(argc,argv)
int argc;
char **argv;
{

register int fildes;
gescape_arg arg1, arg2;

/*  Open device from path and driver name passed in command line */
fildes = gopen (argv[1],OUTDEV,argv[2],INIT);

/* Lock the device, turn semaphore off */
arg1.i[0] = 0;
gescape (fildes, R_LOCK_DEVICE,&arg1,&arg2);      /* Lock */
gescape (fildes, SWITCH_SEMAPHORE,&arg1,&arg2);  /* Semaphore off */

Do graphics operation here. Remember, no "printf", no forks to the device ...
/* Turn on semaphore, unlock the device */
arg1.i[0] = 1;
gescape (fildes, SWITCH_SEMAPHORE,&arg1,&arg2);  /* semaphore on */
gescape (fildes, R_UNLOCK_DEVICE,&arg1,&arg2);   /* Unlock */

/* Close the device */
gclose(fildes);
}
```

R_OFFSCREEN_ALLOC

The ⟨*op*⟩ parameter is `R_OFFSCREEN_ALLOC`.

This `gescape` allows you to use offscreen frame buffer memory in a way that cooperates with offscreen use by Starbase and Windows/9000. Starbase and HP Windows/9000 use offscreen frame buffer memory for storage of raster sprites and characters. You may wish to have a part of offscreen memory allocated for your own personal use. Using this `gescape` will allow you to allocate a portion of offscreen memory for personal usage and will not interfere with Starbase or HP Window/9000 storage. A related `gescape` is `R_OFFSCREEN_FREE`.

The `arg1` parameter contains two integers, specifying the x and y sizes (in pixels) of the rectangular area needed.

The `arg2` parameter returns four integers:

- a success flag (`TRUE` if the allocation was successful and `FALSE` otherwise).
- the raw device coordinates of the allocated rectangle if the allocation was successful.
- the number of pixels to increment from the end of one row in the rectangle to the beginning of the next.

Raw device coordinates are returned even if the request is via a window device file designator. The allocation will fail (return `FALSE` in `arg2[0]`) if there is not a rectangle of the requested size available.

Remember that offscreen memory is used by the driver for raster cursors and fill patterns and also by the Windows/9000 system for the window sprite and raster font optimization. Please read more about the uses of offscreen memory in the appropriate device driver chapter.

On HP 98730 workstations, an alignment factor for x and y can be specified in `arg1[3]` and `arg1[4]` respectively (`arg1[2]` is reserved for future use). The effect of the alignment factor is such that the location modulo for the alignment factor is zero. For example: specifying an x alignment factor of 2 and a y alignment factor of 4, results in an x location on an even boundary (that is, 0, 2, 4, 6, 8, ...) and a y location on a boundary divisible by four (that is, 0, 4, 8, 16, 32, ...). Specifying zero for alignment factors results in no alignment being done.

The following example attempts to allocate a 128×64 pixel rectangle in offscreen frame buffer memory.

C Syntax

```
/* gescape_arg is typedef defined in starbase.c.h */

gescape_arg arg1, arg2;

arg1.i[0]=128;
arg1.i[1]=64;
gescape(fildes,R_OFFSCREEN_ALLOC,&arg1,&arg2);

if (arg2.i[0])
{
   /* allocation successful */
   printf ("OK. Location is %d %d, skipcount is %d.\n",
              arg2.i[1], arg2.i[2], arg2.i[3]);
}
else
{
   /* allocation failed */
   printf ("Oh, well.\n");
}
```

FORTRAN77 Syntax

```
integer*4 arg1(64),arg2(64)
arg1(1)=128
arg1(2)=64
call gescape(fildes,R_OFFSCREEN_ALLOC,arg1,arg2)

if (arg2(1) .eq. TRUE) then
    write *, "OK. Location is ",arg2(2), arg2(3),"."
    write *, "Skipcount is ",arg2(4)"."
else
    write *, "Oh, well."
endif
```

Pascal Syntax

```
{gescape_arg is defined in starbase.p1.h}

var
   arg1,arg2:gescape_arg;

begin
   arg1.i[1] := 128;
   arg1.i[2] := 64;
   gescape(fildes,R_OFFSCREEN_ALLOC,arg1,arg2);

   if arg2.i[1] = 1
       writeln ('OK. Location is ',arg2.i[2],' ',arg2.i[3],'.')
       writeln ('Skipcount is ',arg2.i[4],'.')
   else
       writeln ('Oh, well.');
```

R_OFFSCREEN_FREE

The ⟨*op*⟩ parameter is R_OFFSCREEN_FREE.

This gescape allows you to free offscreen frame buffer memory that has been previously allocated by gescape R_OFFSCREEN_ALLOC.

The arg1 parameter contains two integers, specifying the x and y raw device coordinates of the upper left corner of the rectangular area to be freed.

The arg2 parameter returns one integer; a success flag, TRUE, if the deallocation was successful, and FALSE if otherwise.

The deallocation will fail if the coordinates given do not specify the corner of a previously allocated rectangle. Please read more about the uses of offscreen memory in the appropriate device driver chapter.

The following example deallocates a rectangle in offscreen frame buffer memory at x=1280, y=512.

C Syntax

```
/* gescape_arg is typedef defined in starbase.c.h */

gescape_arg arg1, arg2;

arg1.i[0]=1280;
arg1.i[1]=512;
gescape(fildes,R_OFFSCREEN_FREE,&arg1,&arg2);

if (arg2.i[0])
{
   /* deallocation successful */
   printf ("OK. All gone.\n");
}
else
{
   /* allocation failed */
   printf ("Oh, well.\n");
}
```

FORTRAN77 Syntax

```
integer*4 arg1(64),arg2(64)
 arg1(1)=1280
 arg1(2)=512
 call gescape(fildes,R_OFFSCREEN_FREE,arg1,arg2)

 if (arg2(1) .eq. TRUE) then
     write *, "OK. All gone."
 else
     write *, "Oh, well."
 endif
```

Pascal Syntax

```
{gescape_arg is defined in starbase.p1.h}

var
   arg1,arg2:gescape_arg;

begin
   arg1.i[1] := 1280;
   arg1.i[2] := 512;
   gescape(fildes,R_OFFSCREEN_FREE,arg1,arg2);

   if arg2.i[1] = 1
       writeln ('OK. All gone.')
   else
       writeln ('Oh, well.');
```

R_OV_ECHO_COLORS

The ⟨*op*⟩ parameter is `R_OV_ECHO_COLORS`.

The HP 98720, HP 98721, HP 98730, and HP 98731 can be equipped with 4-overlay planes of frame buffer memory for nondestructive alpha, cursors, or graphics. These overlay planes have their own unique color map, separate from the color map used for the graphics planes.

HP 98720 and HP 98721

The color map for this system consists of sixteen 4-bit entries. These four bits correspond to transparent, red, green, and blue (trgb) in order of MSB to LSB. If the transparent bit (the MSB) is set to zero, the pixel color will be the color of the graphics planes "behind" the overlay planes. If the transparent bit is set to one, the pixel color is forced to the color specified by the red, green, and blue bits in the color map entry. Thus, pixels in the overlay planes can be any combination of the seven primary colors or transparent.

This gescape allows you to specify the color map entries which are used for overlay cursors. As with the graphics planes, the overlay planes actually have two hardware color maps which alternate every 133ms. Therefore, two colors can be specified causing the cursor to blink between the two.

HP 98730 and HP 98731

The color map for this system consists of 16 entries. Each of these entries contains eight bits of red, eight bits of green, eight bits of blue, and a transparency bit. If the transparent bit is set to zero, the pixel color will be the color of the graphics planes "behind" the overlay planes. If the transparent bit is set to one, the pixel color is forced to the color specified by the red, green, and blue bits in the color map entry. Thus, pixels in the overlay planes can be any of the seven primary colors or transparent.

This gescape is provided for backwards compatibility for applications written for the HP 98720 product. This gescape allows you to specify the color of cursors in the fourth overlay plane using only one bit for the red, green, and blue. As with the graphics planes, the overlay planes actually have two hardware color maps which alternate every 133ms. Therefore, two colors are specified causing the cursor to blink between the two. The colors are specified with an 8-bit field

passed in arg1. The upper four bits specify trgb for the primary color map, and the lower four bits specify trgb for the secondary color map.

This gescape causes the color map to be initialized in such a way that the cursor will only be seen in areas of transparency. Therefore, even though the cursor is in the fourth overlay plane, it appears to be in the image planes behind the overlay planes. If another process opened to the overlay planes defines another transparent entry using the R_TRANSPARENCY_INDEX gescape, calling this gescape will cause the color map to be updated so that the cursor will also be seen in this new region of transparency. If this gescape is not called, after defining a new transparency entry in the overlay planes, the cursor for the image planes will not be seen in the regions of the new transparency index.

Refer to the gescape R_ECHO_FG_BG_COLORS for defining overlay cursor colors using the full eight bits of red, green, and blue in the overlay color map.

The arg1 parameter specifies these colors using an 8-bit. The upper four bits specify trgb for the primary color map, and the lower four bits specify trgb for the secondary color map.

The arg2 parameter is ignored.

The program segments below show how to use this gescape to blink overlay cursors between white and transparent.

C Syntax

```
/* gescape_arg is typedef defined in starbase.c.h */

gescape_arg arg1, arg2;

arg1.i[0]=15;
gescape(fildes,R_OV_ECHO_COLORS,&arg1,&arg2);
```

FORTRAN77 Syntax

```
integer*4 arg1(64),arg2(64)
arg1(1)=15
call gescape(fildes,R_OV_ECHO_COLORS,arg1,arg2)
```

Pascal Syntax

```
{gescape_arg is defined in starbase.p1.h}

var
   arg1,arg2:gescape_arg;

begin
   arg1.i[1] := 15;
   gescape(fildes,R_OV_ECHO_COLORS,arg1,arg2);
```

R_OVERLAY_ECHO

The ⟨*op*⟩ parameter is `R_OVERLAY_ECHO`.

This `gescape` allows you to select whether graphics cursors will be located in the graphics or overlay planes. Placing cursors in the overlay planes may significantly improve driver performance while cursors are turned on because the driver does not have to "pick up" the cursor to draw. Images created in the overlay planes do not affect images in the graphics planes.

You can specify the location of raster and non-raster cursors separately.

The `arg1` parameter contains two flags; the first flag specifies the location of raster cursors, the second specifies the location of non-raster cursors. If the flag is `TRUE`, the corresponding cursors will be echoed in the overlay plane, if `FALSE`, the corresponding cursors will be echoed in the graphics planes.

The `arg2` parameter is ignored.

You must call `define_raster_echo to actually move the raster echo into the overlay planes.` See the `DEFINE_RASTER_ECHO(3G)` entry in the *Starbase Reference* for more information.

HP 98550 and HP 98556

The HP 319C, HP 98549A, and HP 98550A displays may be opened in configurations that provide 2-overlay planes in addition to 4- or 8-image planes.

If the overlay planes are simultaneously accessed through another `gopen`, there is no safeguard to prevent unwanted interactions.

Non-raster and raster cursors may be placed in either the overlay or the image planes. Both default to the planes specified by the special device file used with the `gopen` procedure.

This `gescape` has no effect when the fildes used corresponds to `gopen` of the overlay planes. Note that an overlay cursor may not appear as expected if the overlay color map has not been initialized.

This `gescape` has no effect on the HP 98548A display.

HP 98720 and HP 98721

The HP 98720 and HP 98721 can be equipped with four overlay planes of frame buffer memory for nondestructive alpha, cursors, or graphics.

Only one overlay plane can be used for cursors, so overlay cursors must be monochrome. Review the `R_OV_ECHO_COLORS gescape` described in this appendix for details on overlay cursor colors. Raster cursors in the graphics planes may be any combination of available colors.

Since there is little advantage to having non-raster cursors in the graphics planes and performance suffers, non-raster cursors default to the overlay plane. Since it may be very desirable to have multi-colored raster cursors, these default to the graphics planes.

HP 98730

The HP 98730 can be equipped with four overlay planes of frame buffer memory for nondestructive alpha, cursors, or graphics. This `gescape` allows you to select whether graphics cursors will be located in the graphics planes or the fourth overlay plane (when opened to the graphics planes). Placing cursors in the fourth overlay plane can significantly improve driver performance while cursors are turned on. Note that the HP 98731 Device Driver always overlays cursors. Only one overlay plane can be used for cursors, so overlay cursors must be monochrome. Review the `R_OV_ECHO_COLORS gescape` described in this appendix for details on overlay cursor colors. Raster cursors in the graphics planes may be any combination of available colors. Performance will be reduced, however, since each time the display is altered, it is necessary to "pick up" the cursor, make the alteration, and put down the cursor.

Since there is little advantage to having non-raster cursors in the graphics planes, and performance suffers, non-raster software cursors default to the overlay plane. Since it may be very desirable to have multi-colored raster cursors, these default to the graphics planes.

Cursors can only be put in the fourth overlay plane when there is a fourth overlay plane. If another process opens all overlay planes, this `gescape` will not allow placing cursors in the fourth overlay plane. In an X window, overlay plane cursors may be available even when some other process has all the overlay planes open. See the *Starbase Programming with X11* manual for more information.

This `gescape` is a `no-op` if the hardware cursor is being used.

C Syntax

```
/* gescape_arg is typedef defined in starbase.c.h */

gescape_arg arg1, arg2;

arg1.i[0]=TRUE;
arg1.i[1]=TRUE;
gescape(fildes,R_OVERLAY_ECHO,&arg1,&arg2);
```

FORTRAN77 Syntax

```
integer*4 arg1(64),arg2(64)
arg1(1)=TRUE
arg1(2)=TRUE
call gescape(fildes,R_OVERLAY_ECHO,arg1,arg2)
```

Pascal Syntax

```
{gescape_arg is defined in starbase.p1.h}

var
   arg1,arg2:gescape_arg;

begin
   arg1.i[1] := 1;
   arg1.i[2] := 1;
   gescape(fildes,R_OVERLAY_ECHO,arg1,arg2);
```

R_TRANSPARENCY_INDEX

The ⟨*op*⟩ parameter is `R_TRANSPARENCY_INDEX`.

HP 98720 and HP 98721

The HP 98720 and HP 98721 can be equipped with four overlay planes of frame buffer memory for nondestructive alpha, cursors, or graphics. These overlay planes have their own unique color map, separate from the color map used for the graphics planes. This color map consists of sixteen 4-bit entries. These four bits correspond to transparent, red, green, and blue (trgb) in order of MSB to LSB. If the transparent bit (the MSB) is set to zero, the pixel color will be the color of the graphics planes "behind" the overlay planes. If the transparent bit is set to one, the pixel color is forced to the color specified by the red, green, and blue bits in the color map entry. Thus, pixels in the overlay planes can be any combination of the seven primary colors or transparent.

If a graphics driver has been opened to the overlay planes, this **gescape** can be used to create a transparent color entry in the color map. When the color maps are initialized, all entries have the transparency bit set to one. This **gescape** clears that bit for the specified color index. If the entry is updated, as in a call to `define_color_table`, the transparency bit is set back to one.

Note that this **gescape** will have no effect if the graphics driver has been opened to the graphics planes rather than the overlay planes.

HP 98730 and HP 98731

The HP 98730 and HP 98731 come equipped with four overlay planes of frame buffer memory for non-destructive alpha, cursors, or graphics. These overlay planes have their own unique color map, separate from the color map used for the graphics planes. This color map consists of sixteen 24-bit color entries and sixteen transparent entries. Each color map entry has eight bits for red, eight bits for green, and eight bits for blue. For each color entry there is a transparency bit. If this bit is zero, the pixel color in the overlay plane is blended with the pixel color in the graphics planes "behind" the overlay planes. If the transparency bit is set to one, the pixel color in the overlay plane is forced to the color specified by the red, green, and blue bits in the overlay color map.

If the graphics driver has been opened to the overlay planes, this **gescape** can be used to create a transparent color entry in the overlay color map. When the color

maps are initialized, all entries have their transparency bits set to one. (This is only true if the environment variable SB_OV_SEE_THRU_INDEX is set to -1. Refer to the HP 98730 and HP 98731 driver sections for details.) This gescape can be used to set a color map entry to transparent (that is, the color black for a pixel is blended with the pixel color in the image planes behind the overlay planes). If the entry is updated, as in a call to define_color_table, the transparency bit is set back to one.

Note that this gescape will have no effect if the graphics driver has been opened to the graphics planes rather than the overlay planes.

The arg1 parameter contains to the transparency index.

The arg2 parameter is ignored.

The examples below demonstrate setting index 0 to transparent.

C Syntax

```
/* gescape_arg is typedef defined in starbase.c.h */

gescape_arg arg1, arg2;

arg1.i[0]=0;
gescape(fildes,R_TRANSPARENCY_INDEX,&arg1,&arg2);
```

FORTRAN77 Syntax

```
integer arg1(64),arg2(64)
arg1(1)=0
call gescape(fildes,R_TRANSPARENCY_INDEX,arg1,arg2)
```

Pascal Syntax

```
{gescape_arg is defined in starbase.p1.h}

var
   arg1,arg2:gescape_arg;

begin
   arg1.i[1] := 0;
   gescape(fildes,R_TRANSPARENCY_INDEX,arg1,arg2);
```

R_UNLOCK_DEVICE

The ⟨*op*⟩ parameter is R_UNLOCK_DEVICE.

This procedure unlocks the device associated with the specified file descriptor (fildes).

This procedure should be called prior to turning semaphores on if R_LOCK_DEVICE was used to lock the device. See R_LOCK_DEVICE for an example program.

Both the arg1 and arg2 parameters are ignored.

The syntax of this procedure is the same for both a window device and the raw device. The lock and unlock gescape functions are useful when semaphores are turned off, and the program needs use of the display.

When fildes is associated with a window, arg1.i[0] is significant. If arg1.i[0] != 0, the window system sprite will be restored (should one be visible). Many unlocks can be done in a row as long as the same number of locks have already been done. Regardless of arg1.i[0], the last unlock always causes the sprite to be restored on the display.

C Syntax

```
/* gescape_arg is typedef defined in starbase.c.h */

gescape_arg arg1, arg2;

gescape(fildes,R_UNLOCK_DEVICE,&arg1,&arg2);
```

FORTRAN77 Syntax

```
integer*4 arg1(64),arg2(64)
call gescape(fildes,R_UNLOCK_DEVICE,arg1,arg2)
```

Pascal Syntax

```
{gescape_arg is defined in starbase.p1.h}

var
   arg1,arg2 : gescape_arg;

begin
   gescape(fildes,R_UNLOCK_DEVICE,arg1,arg2);
```

READ_COLOR_MAP

The ⟨*op*⟩ parameter is READ_COLOR_MAP.

This gescape copies the device's hardware color map into the software color map associated with the file descriptor. The software color map is used by the Starbase library for dither calculations, color specification, and inquires.

This gescape is ignored when the display is black and white.

This gescape is ignored for terminals other than the HP 2397 and when output is spooled for terminals. READ_COLOR_MAP can be used to get the color map definition as defined by the hardware. The software color map and hardware color map can differ when multiple processes are changing the color table. Another time that this gescape is useful is when you wish to allow a process to function without changing the actual color map. To do this, read the current hardware color map state after opening a graphics device with the gopen mode set without INIT. See the *Starbase Programming with X11* manual for information on using this gescape in an X11 window.

Both the arg1 and arg2 parameters are ignored.

C Syntax

```
/* gescape_arg is typedef defined in starbase.c.h */

gescape_arg arg1, arg2;

gescape(fildes,READ_COLOR_MAP,&arg1,&arg2);
```

FORTRAN77 Syntax

```
integer*4 arg1(64),arg2(64)
call gescape(fildes,READ_COLOR_MAP,arg1,arg2)
```

Pascal Syntax

```
{gescape_arg is defined in starbase.p1.h}

var
   arg1,arg2 : gescape_arg;

begin
   gescape(fildes,READ_COLOR_MAP,arg1,arg2);
```

SET_BANK_CMAP

The ⟨*op*⟩ parameter is SET_BANK_CMAP.

This gescape allows you to select individual color maps for separate frame buffer banks. The HP 98730 device supports up to three separate frame buffer banks of eight planes each. Each can have its own unique color map. By default, all color maps are loaded identically. This gescape allows them to be different. This is primarily intended for use when frame buffer outputs are being blended (see the gescape IMAGE_BLEND). When blending, this function allows you to vary the contribution of each bank with define_color_table. For example, if a given bank's color map entries were smoothly zeroed out, the displayed image from that bank would smoothly fade out.

The arg1 parameter points to the argument list for this function. It takes one argument: an integer specifying which bank is being selected. Allowable values are zero through two.

The arg2 parameter is ignored.

The bank selected by this gescape will have its color map installed for subsequent Starbase calls. This means that calls to define_color_table will affect only the installed color map. Also, functions which search the color map will use the newly installed color map. For example: in CMAP_NORMAL mode fill_color may search color map entries to form a dither cell or find the closest match. The color map it searches will be the one installed with this gescape.

The examples below demonstrate changing the color map for bank one to the values in the array "colors".

C Syntax

```
/* gescape_arg is typedef defined in starbase.c.h */

gescape_arg arg1, arg2;
float colors[256][3];

arg1.i[0] = 1;      /* choose bank 1 */
gescape(fildes,SET_BANK_CMAP,&arg1,&arg2);
define_color_table(fildes,0,256,colors);
                /* Update entire cmap for bank 1 */
```

FORTRAN77 Syntax

```
int arg1(1),arg2(1)
     real colors(3,256)

arg1(1)=1
call gescape(fildes,SET_BANK_CMAP,arg1,arg2)
call define_color_table(fildes,0,256,colors)
```

Pascal Syntax

```
   {gescape_arg is defined in starbase.p1.h}

   type rgb_color=array[1..3]of real;
   var
     arg1,arg2:gescape_arg;
     colors: array[0..256] of rgb_color;
   begin
     arg1.i[1]:= 1;
     gescape(fildes,SET_BANK_CMAP,arg1,arg2);
     define_color_table (fildes,0,256,colors);
end
```

SWITCH_SEMAPHORE

The ⟨*op*⟩ parameter is SWITCH_SEMAPHORE.

Semaphore operations prevent interference between multiple processes accessing the same device. Semaphore operations are normally on. See R_LOCK_DEVICE for an example of how this control is used for multiple processes accessing the same device.

If only a single process is accessing a device, you can significantly increase speed by turning the semaphore operations off.

The TRACK procedure will also turn the semaphore operations on. Do not turn the semaphore operations off when the output device has an asynchronous process tracking to it.

The arg1 parameter switches the semaphore operations on (if TRUE (1)) and off (if FALSE (0)).

The arg2 parameter is ignored.

If you want to hold the display for a long time and run with the speed improvement of not checking the semaphore, the following process is suggested:

1. Lock down device to guarantee that the process is the sole owner of the display. See the gescape function R_LOCK_DEVICE.
2. If the device is a window, you must insure that the window is unobscured. See the gescape function R_GET_WINDOW_INFO—this gescape will also work to the raw device; in this case it always says it's "unobscured". If the window is obscured, you must unlock the device. See the gescape function R_UNLOCK_DEVICE and try again later.
3. Do the semaphore switch to improve performance slightly.
4. Do whatever Starbase operations are desired, for as long as desired.
5. Do the semaphore switch to re-enable lock/unlock operations with semaphores. If the device is a window, this also re-enables output to obscured windows.

6. Unlock the device using the gescape function R_UNLOCK_DEVICE to allow other processes to access the display. Windows/9000 or another graphics application are examples of other processes that will block until this gescape is finished.

See the R_GET_WINDOW_INFO gescape for a C program example.

The following examples switch semaphore operations on.

C Syntax

```
/* gescape_arg is typedef defined in starbase.c.h */

gescape_arg arg1, arg2;

arg1.i[0]=TRUE;
gescape(fildes,SWITCH_SEMAPHORE,&arg1,&arg2);
```

FORTRAN77 Syntax

```
integer*4 arg1(64),arg2(64)
arg1(1)=TRUE
call gescape(fildes,SWITCH_SEMAPHORE,arg1,arg2)
```

Pascal Syntax

```
{gescape_arg is defined in starbase.p1.h}

var
   arg1,arg2 : gescape_arg;

begin
   arg1.i[1]:=1;
   gescape(fildes,SWITCH_SEMAPHORE,arg1,arg2);
```

TRIGGER_ON_RELEASE

The ⟨*op*⟩ parameter is TRIGGER_ON_RELEASE.

The default trigger is started when a button is pressed. This allows events to be triggered when a button is released.

The arg1 and arg2 parameters are ignored.

C Syntax Example

```
/* gescape_arg is type defined in starbase.c.h */
gescape_arg arg1, arg2;
gescape(fildes,TRIGGER_ON_RELEASE,&arg1,&arg2);
```

FORTRAN77 Syntax Example

```
integer*4 arg1(64),arg2(64)

call gescape(fildes,TRIGGER_ON_RELEASE,arg1,arg2)
```

Pascal Syntax Example

```
{gescape_arg is defined in starbase.p1.h}
var
   arg1, arg2 : gescape_arg;
begin
   gescape(fildes,TRIGGER_ON_RELEASE,arg1,arg2);
```

TRANSPARENCY

The ⟨*op*⟩ parameter is `TRANSPARENCY`

This `gescape` allows you to define a "screen door" transparency pattern for use with polygon rendering. You may define a pattern that disables writes to any pixels within a 4×4 cell. This cell is duplicated over the entire screen.

Pass in a bit mask where a "1" means the corresponding pixel is write enabled and a "0" is write disabled. Table A-8 shows the 2 byte pattern passed in, and table A-9 shows how that pattern is turned into a 4×4 dither cell.

This `gescape` will set the same pattern for both front and back facing polygons. To define different patterns for front facing polygons and back facing polygons, use the `POLYGON_TRANSPARENCY gescape` (HP 98731 only).

The `arg1` parameter contains the mask.

The `arg2` parameter is ignored.

Table GESC-8.

15	...	2	1	0

Table GESC-9.

0	1	2	3
4	5	6	7
8	9	10	11
12	13	14	15

The examples will produce a green square with a 50 percent transparent red rectangle in front. Remember to reset the transparency to opaque when done.

C Syntax

```
/* gescape_arg is typedef defined in starbase.c.h */

gescape_arg arg1, arg2;
```

```
fill_color(fildes,0.0,1.0,0.0);
rectangle(fildes,0.25,0.25,0.75,0.75);
arg1.i[0] = 0xAAAA;
gescape(fildes,TRANSPARENCY,&arg1,&arg2);
fill_color(fildes,1.0,0.0,0.0);
rectangle(fildes,0.0,0.25,1.0,0.75);
arg1.i[0] = 0xFFFF;
gescape(fildes,TRANSPARENCY,&arg1,&arg2);
```

FORTRAN77 Syntax

```
      integer*4 arg1(64),arg2(64),pattern(2)
      data pattern/z'0A0A0A0A',
C                 z'0F0F0F0F'/
      fill_color(fildes,0.0,1.0,0.0);
      rectangle(fildes,0.25,0.25,0.75,0.75);
      arg1(1)=pattern(1)
      call gescape(fildes,TRANSPARENCY,arg1,arg2)
      fill_color(fildes,1.0,0.0,0.0);
      rectangle(fildes,0.0,0.25,1.0,0.75);
      arg1(1)=pattern(2)
      call gescape(fildes,TRANSPARENCY,arg1,arg2)
```

Pascal Syntax

```
{gescape_arg is defined in starbase.p1.h}

var
   arg1,arg2:gescape_arg;

begin
     fill_color(fildes,0.0,1.0,0.0);
     rectangle(fildes,0.25,0.25,0.75,0.75);
     arg1.i[1] :=hex('AAAA');
     gescape(fildes,TRANSPARENCY,arg1,arg2);
     fill_color(fildes,1.0,0.0,0.0);
     rectangle(fildes,0.0,0.25,1.0,0.75);
     arg1.i[1] := hex('FFFF');
     gescape(fildes,TRANSPARENCY,arg1,arg2);
```

ZWRITE_ENABLE

This gescape was designed specifically to allow the creation of three-dimensional cursors in the overlay planes. To accomplish this objective, you need to draw a primitive in the overlay planes to use the same Z-buffer used to draw the object in the image planes. To get a three-dimensional cursor effect, this gescape allows primitives to be rendered using the Z-buffer information, but the primitives do not modify the Z-buffer in any way. This gescape may be used in conjunction with the ZSTATE_SAVE and ZSTATE_RESTORE gescapes to accomplish three-dimensional cursors in the overlay planes.

Devices with dedicated Z-buffers do not need to use ZSTATE_SAVE and ZSTATE_RESTORE. However, if they are used, they will have no detrimental effects.

Devices which support analog blending of frame buffer outputs can be used to achieve three-dimensional cursor effects without using the overlay planes, because different frame buffer banks may be used for the cursors and the image. See the IMAGE_BLEND gescape for more information on blending.

The gescape ZWRITE_ENABLE looks at arg1[0] to determine whether to enable (arg1[0]!=0) or disable (arg1[0]=0) the Z-buffer for primitive modification. This has no effect on zbuffer_switch which will clear the Z-buffer.

The examples below will disable the zbuffer from primitive modification.

C Syntax

```
/* gescape_arg is typedef defined in starbase.c.h */

gescape_arg arg1, arg2;

arg1.i[0]=0;
gescape(fildes,ZWRITE_ENABLE,&arg1,&arg2);
```

FORTRAN77 Syntax

```
integer*4 arg1(64),arg2(64)

arg1(1)=0
call gescape(fildes,ZWRITE_ENABLE,arg1,arg2)
```

Pascal Syntax

```
{gescape_arg is defined in starbase.p1.h}

var
   arg1,arg2:gescape_arg;

begin
   arg1.i[1]:=0
   gescape(fildes,ZWRITE_ENABLE,arg1,arg2);
```

Win an HP Calculator!

Your comments and suggestions help us determine how well we meet your needs. **Returning this card with your name and address enters you into a quarterly drawing for an HP calculator***.

Starbase Device Drivers
Library 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 month
__ Less than 1 year
__ 1 to 2 years
__ More than 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: _____

*Offer expires June 1990. (98592-90018 E0989)

Please Tape Here

Please print or type your name and address.

Name: _____

Company: _____

Address: _____

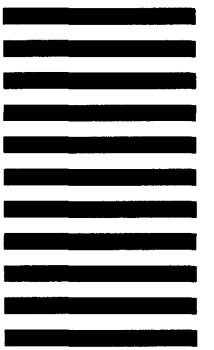
City, State, Zip: _____

Telephone: _____

Additional Comments: _____

Starbase Device Drivers Library Manual
HP Part Number 98592-90018
E0989

NO POSTAGE
NECESSARY
IF MAILED
IN THE
UNITED STATES



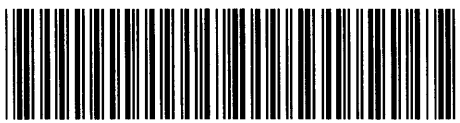
BUSINESS REPLY MAIL

FIRST CLASS PERMIT NO. 37 LOVELAND, COLORADO

POSTAGE WILL BE PAID BY ADDRESSEE

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

HP Part Number
98592-90018
Microfiche No. 98592-99018
Printed in U.S.A. E0989



98592-90601
For Internal Use Only