HP-UX Starbase Device Drivers Manual

Volume 2

HP 9000 Series 700 Computers

HEWLETT PACKARD

HP Part No. B2355-90047
Printed in USA April, 1993

Edition 1

Notices

Printing History

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

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

April 1993 . . . Edition 1. This manual is valid for HP-UX release 9.0 on all HP 9000 Series 700 Computers. This edition of the manual includes new HP VMX information as well as manual corrections.

This manual includes some Series 300/400/800 Starbase information; however, for revision 9.0 Starbase information on Series 300/400/800 computers, you should read the *HP-UX Starbase Device Drivers Manual* part number B2355-90019.

FINAL TRIM SIZE : 7.5 in x 9.0 in

Contents

Tables

16 The Starbase Memory Driver

Device Description

The Starbase Memory Driver (SMD) permits the user to treat memory like a frame-buffer device and direct Starbase operations to it. The SMD can be used for quick pop-up menus from offscreen, shadow buffering (creating images offscreen and then move rapidly to on-screen), etc. See the chapter "The Starbase Memory Driver" in the *Starbase Graphics Techniques* manual for further information on what the SMD is and how to use it.

The SMD driver supports three modes:

- `SMDpixel` mode (pixel-major packing format with one bank)
- `SMDpixel3` mode (pixel-major packing format with three banks)
- `SMDplane` mode (plane-major packing format with 1, 2, or 3 banks)

Setting Up the Device

Switch Settings

Switch settings are not applicable to a memory-resident frame buffer.

Special Device Files (mknod)

No special device file need be created when using the SMD, since the `gopen` path name used is `/dev/null`.

Linking the Driver

Shared Libraries

`SMDpixel` and `SMDpixel3` is the shared library file `libddSMDpix.sl` in the `/usr/lib` directory. The shared library file will be explicitly loaded at run time by compiling and linking with the starbase shared library `/usr/lib/libsb.sl`, or by using the `-l` option `-lsb`.

Examples

To compile and link a C program for use with the shared library driver, use:

```
cc example.c -I/usr/include/X11R5/x11 -L/usr/lib/X11R5\
-lXwindow -lsb -lXhp11 -lX11 -ldld -lm -o example
```

or with FORTRAN use,

```
F77 example.f -Wl,-L/usr/lib/X11R5 -lXwindow -lsb\
-lXhp11 -lX11 -ldld -lm  -o example
```

or with Pascal use,

```
pc example.p  -Wl,-L/usr/lib/X11R5 -lXwindow -lsb\
-lXhp11 -lX11 -ldld -lm -o example
```

For details, see the discussion of the `gopen` procedure in the section *To Open and Initialize the Device* in this chapter.

Archive Libraries

`SMDpixel` and `SMpixel3` is the archive file `libddSMDpix.a` in the directory `/usr/lib`. This device driver may be linked to a program using the absolute path name `/usr/lib/libddSMDpix.a`, an appropriate relative path name, or by using the `-l` option as in `-lddSMDpix` with the `LDOPTS` environmental variable set to `-a archive`.

The reason for using the `LDOPTS` environmental variable is that the `-l` option will look for a shared library driver first and then look for the archive driver if shared was not found. By exporting the `LDOPTS` variable as specified above, the `-l` option will only look for archive drivers. For more information, refer to the *Programming on HP-UX* manual on linking shared or archive libraries.

Assuming you are using `ksh`(1), to compile and link a C program for use with this driver, use:

```
export LDOPTS="-a archive"
```

and then:

```
cc example.c -lddSMDpix -L/usr/lib/X11R5 -lXwindow\
-lsb1 -lsb2 -lXhp11 -lX11 -lm  -o example
```

or for FORTRAN, use:

```
F77 example.f -lddSMDpix -Wl,-L/usr/lib/X11R5 -lXwindow\
 -lsb1 -lsb2 -lXhp11 -lX11 -o example
```

or for Pascal, use:

```
pc example.p -lddSMDpix -Wl,-L/usr/lib/X11R5 -lXwindow\
 -lsb1 -lsb2 -lXhp11 -lX11 -o example
```

`SMDplane` is the file `libddSMDpln.a` in the directory `/usr/lib`. This driver is linked the same way as `SMDpixel`.

If you are using raster fonts, you must also link in the `libfontm.a` library.

Device Initialization

Parameters for gopen

The `gopen` procedure has four parameters: Path, Kind, Driver and Mode.

```
   fildes = gopen(Path, Kind, Driver, Mode);
```

Path	Always `/dev/null` when using the Starbase memory driver.
Kind	Indicates the I/O characteristics of the device. This parameter must be `OUTDEV` for this driver.
Driver	The character representation of the driver type. This must be either `SMDpixel`, `SMDpixel3`, or `SMDplane`. For example:

`"SMDpixel"`	*for C.*
`'SMDpixel'//char(0)`	*for FORTRAN77.*
`'SMDpixel'`	*for Pascal.*

`SMDpixel` is for byte-per-pixel with a depth of 8, and `SMDpixel3` is for byte-per-pixel and three banks, giving a depth of 24.

`SMDplane` is for bit-per-pixel with a depth of up to 24 planes.

Mode — The mode control word consisting of several flag bits which can be *or* ed together. Listed below are those flag bits which have no affect for this driver and those which have device-dependent actions. Flags not discussed below operate as defined by the `gopen` procedure.

The SMD supports mode values of the `RESET_DEVICE`, `INIT`, `THREE_D`, and `MODEL_XFORM` flags. For `MODEL_XFORM`, shading and hidden-surface removal are not supported. However, opening in `MODEL_XFORM` mode affects how matrix stack and transformation routines are performed.

For all modes, the software color map is automatically initialized.

- The `SPOOLED` flag bit causes an error for this driver and cannot spool memory buffers.
- The following flag bits have device dependent actions:

- `0`—open the device, but defer memory buffer allocation until explicitly requested through `gescape` or until the first graphics primitive is called.
- `INIT` and `RESET_DEVICE`—open and initialize the device as follows:

  1. The memory buffer is allocated.
  2. Clear memory buffer to 0s.
  3. Reset the color map to its default values.

Syntax Examples

To open and initialize an SMD device (`SMDpixel3` may be substituted for `SMDpixel` if you desire a three-bank memory buffer; `SMDplane` may be substituted for bit-per-pixel memory applications.):

For C Programs:

```
fildes = gopen("/dev/null", OUTDEV, "SMDpixel", INIT);
```

For FORTRAN77 Programs:

```
fildes = gopen('/dev/null'//char(0), OUTDEV, 'SMDpixel'//char(0), INIT)
```

For Pascal Programs:

```
fildes = gopen('/dev/null', OUTDEV, 'SMDpixel', INIT);
```

Special Device Characteristics

For device coordinate operations, location (0, 0) is the upper-left corner of the memory buffer with X-axis values increasing to the right and Y-axis values increasing down. The lower-right corner of the buffer is therefore xmax, ymax, where xmax and ymax are 1 less than the X-size and Y-size specified for the memory buffer. The size can be set through calling the `gescape` procedure `SMD_DEFINE_XY`.

Starbase Functionality

Commands Not Supported

This section notes which standard Starbase capabilities are *not* supported by the SMD:

- An SMD memory buffer is an output-only device. Thus, the following Starbase input-related calls are not supported (no action is taken if they are called):

  ```
  await_event              read_locator_event
  define_raster_echo       request_choice
  disable_events           request_locator
  echo_type                sample_choice
  echo_update              set_locator
  enable_events            set_signal
  initiate_request         track
  inquire_request_status   track_off
  read_choice_event
  ```

  A call to `inquire_input_capabilities` indicates that there are no input capabilities.

- The SMD's memory is never visible; you can never see the image with your eyes. Thus, these two visibility-related calls are ignored for the SMD.

  ```
  await_retrace            display_enable
  ```

- The SMD does not emulate features provided by device hardware. For example, the Z-buffer hidden-surface removal and shading that can be done by the transform engine drivers are not supported. Explicitly, the functions not supported are:

```
alpha_transparency             light_attenuation
backface_control               light_model
bf_alpha_transparency          light_switch
bf_control                     line_endpoint
bf_fill_color                  line_filter
bf_interior_style              perimeter_filter
bf_perimeter_color             set_capping_planes
bf_perimeter_repeat_length     set_model_clip_indicator
bf_perimeter_type              set_model_clip_volume
bf_surface_coefficients        shade_range
bf_surface_model               surface_coefficients
bf_texture_index               surface_model
contour_enable                 texture_index
define_contour_table           texture_viewport
define_texture                 texture_window
define_trimming_curve          viewpoint
deformation_mode               zbuffer_switch
depth_cue
depth_cue_color
depth_cue_range
hidden_surface
interior_style (INT_OUTLINE)
interior_style (INT_POINT)
intline_width
light_ambient
```

Conditionally Supported

Routines which are partially supported are:

`bank_switch`	For `SMDpixel` mode, this call is ignored. For `SMDpixel3` and `SMDplane`, this call is supported.
`shade_mode`	The color map mode may be selected, but shading cannot be turned on

`vertex_format`	The user can call this routine, but the driver does not recognize any extra coordinates.

`with_data`	`partial_polygon_with_data3d`
	`polygon_with_data3d`
	`polyhedron_with_data`
	`polyline_with_data3d`
	`polymarker_with_data3d`
	`quadrilateral_mesh_with_data`
	`triangle_strip_with_data`
	Additional data per vertex will be ignored if not supported by this device. For example, contouring data will be ignored if the device does not support it.

Configuration

- A packing format to emulate the full 1024×400 resolution of the Series 300 medium-resolution frame buffer (driver `3001` is not supported).

Note The `3001` driver normally does vector generation turning on two pixels at a time. This is because its resolution is actually 1024×400, but Starbase treats the device as 512×400. There is a `gescape` operation that lets you treat the `3001` resolution as 1024×400 during `block_write` and `block_read` operations. When using this mode, the driver does not skip every other byte of input (or output).

- The SMD driver does not provide locking and unlocking capabilities that permit shared access to a single memory buffer.
- The HP 98720 driver supports 8 planes, 16 planes, or 24 planes of frame buffer. If using the 16 planes, only 8 planes are displayabled at a time (double-buffered). With 24 planes, you can double-buffer with two sets of 12 planes. Thus, these modes are intended for double-buffering applications. `SMDpixel3`

always emulates a full 24 planes and does not emulate 8 or 16 planes. You can use these 24 planes to double-buffer with either 8 or 12 planes per buffer.

- `SMDplane` emulates exactly the number of planes specified:
  - 1, 3, 4, 6, or 8 planes: 1 bank.
  - 16 planes: 2 banks (8 planes, double-buffered).
  - 24 planes: 3 banks.

Fast Alpha and Font Manager

The SMD supports raster text calls from the "Fast Alpha and Font Manager Libraries" as documented in *Fast Alpha/Font Manager Programmer's Manual*. Since raster fonts consist of one byte per pixel, raster text is written only to the currently selected bank when the SMD is being used in `SMDpixel3` mode. This is similar to the operation of other raster functions like `block_write`.

SMD Errors

The general philosophy of SMD error reporting is that when a Starbase function is invoked which is not supported by SMD a no-op is performed, but no error or warning is issued.

Errors are issued for operations that will not work, for example, input on an output-only device.

Harmless errors (a color map index out of range) cause warnings to be issued.

SMD reports the following errors:

- SMD opened with `SPOOLED`, `OUTINDEV`, or `INDEV` specified.
- The user supplies an address to a block of memory for the memory buffer after the memory buffer has already been allocated. Or the user supplies the `NULL` pointer in the `SMD_SUPPLY_MEM_BUFF` parameter of the `gescape` call.
- The user tries to redefine the depth via `gescape` on an `SMDpixel3` format.
- The user supplies an invalid `gescape` opcode.

- After frame buffer resizing, either by a depth redefinition or X, Y redefinition, the frame buffer is greater than 2^{32}-1 bytes (4 gigabytes). This is the maximum size that a frame buffer can be.
- User tries to redefine the depth for a `SMDpixel` memory buffer beyond 8 planes or some value other than 1,3,4,6, or 8.
- User tries to define the depth for a `SMDplane` memory buffer beyond 24 planes or some value other than 1, 3, 4, 6, 8, 16, or 24.
- User specifies an X or Y value in `R_DEFINE_XY` larger than 2^{15}-1 (32,767).
- The memory buffer could not be allocated. One possible reason is that the size causes the application to exceed its current address space limitation. The maximum amount of memory that can be allocated depends on the amount of swap space available to the system, the maximum data segment size per process in the system, and the number of processes running. The total address space (used by all currently running processes) cannot exceed the amount of swap space available in the system.

  If the SMD is unable to allocate the amount of memory requested, it returns what it can allocate depending on the amount asked for. If the allocation happens via `gescape`, this information is returned in `arg2`. If the allocation happens at `gopen` or at the time of the first graphics primitive, this information is reported to `stderr`.

  The amount SMD claims it can allocate will fluctuate, since it is dependent on the number of other processes running at that time.

  It is your responsibility to size down your application or other processes to be able to get the memory you are requesting. Possible methods of sizing down the application are:
  - Decrease the number of other bitmapped display drivers that are running with the application. Each driver maps the frame buffer into the address space.
  - Decrease how much memory is being requested from the SMD.
  - Reconfigure your system with more swap space or a larger data segment size per process (refer to the system's configuration manual).
  - Decrease the number of other processes running in the system concurrently. This will give more address space to the SMD application program.

Parameters for gescape

Detailed information about `gescape` functions can be found in Appendix A of this manual.

- `R_BIT_MODE`—bit mode (supported by `SMDpixel` and `SMDpixel3`).

Note `R_BIT_MODE` is always true for `SMDplane`.

- `R_BIT_MASK`—bit Mask (supported by `SMDpixel` and `SMDpixel3`).
- `DEF_FILL_PAT`—define fill pattern (supported by `SMDpixel` and `SMDpixel3`).
- `R_GET_FRAME_BUFFER`—get frame buffer pointer (supported by all SMD drivers).
- `R_LINE_TYPE`—define line type

The following `gescape` functions are unique to this driver:

- `SMD_DEFINE_DEPTH`—define memory buffer depth
- `SMD_DEFINE_XY`—define X, Y dimensions
- `SMD_SUPPLY_MEM_BUFF`—supply memory buffer
- `SMD_GET_MEM_REQUIRED`—determining memory requirements
- `SMD_ALLOCATE_MEMORY`—allocate frame buffer

17

The Starbase-on-X11 Device Driver

Device Description

The Starbase-on-X11 (sox11) device driver libraries,`libddsox11.a` and `libddsox11.sl`, allow an application to use Starbase functions within version 11 of the X Window System. The sox11 driver implements the device-dependent Starbase routines by calling the X11 library, Xlib, and may be described as an implementation of Starbase "on top of" X11.

The implementation allows Starbase applications to use the features of version 11 of the X Window System. This includes running applications over the network, where the application runs as a client on one machine while using the X11 display server to perform I/O either on another machine or locally. A Starbase application can use any HP 9000 Series 300/400/700/800 machine running Starbase as its X11 client machine, while the same application may use any accessible hardware running an X11 server to perform all I/O operations. Thus an X11 window serves as a virtual device for Starbase.

Not all Starbase calls (for example, 3D solids-modeling calls) can be translated into Xlib calls because this driver does not support full Starbase functionality. However, since Xlib works over the network between a client and a server, the sox11 driver permits Starbase to work over the network, but with reduced functionality and performance compared to Starbase running with the Starbase display drivers.

Note that the X11 server performing I/O for the application need not necessarily be running on HP equipment, however, differences in behavior can result if such non-HP equipment is used.

Setting Up a sox11 Environment

The sox11 driver can be used on any machine once Starbase and version 11 of the X Window System have been installed. No special installation need be performed, and no special nodes need to be made in order to use the sox11 driver. As long as Starbase and X11 have been installed on the system, applications may run on any accessible display being controlled by an X11 server.

Linking the sox11 Driver with an Application

Shared Libraries

The `/usr/lib` directory contains the shared sox11 Device Driver file named `libddsox11.sl`.

The device driver will be explicitly loaded at run time by compiling and linking with the Starbase shared library `/usr/lib/libsb.sl`, or by using the `-l` option `-lsb`.

Examples

To compile and link a C program for use with the shared library driver, use:

```
cc example.c -I/usr/include/X11R5/x11 -L/usr/lib/X11R5\
-lXwindow -lsb -lXhp11 -lX11 -ldld -lm -o example
```

or with FORTRAN use,

```
F77 example.f -Wl,-L/usr/lib/X11R5 -lXwindow -lsb\
-lXhp11 -lX11 -ldld -lm  -o example
```

or with Pascal use,

```
pc example.p  -Wl,-L/usr/lib/X11R5 -lXwindow -lsb\
-lXhp11 -lX11 -ldld -lm -o example
```

For details, see the discussion of the `gopen` procedure in the section *To Open and Initialize the Device* in this chapter.

To use the sox11 device driver, two others libraries, Xlib `-lX11` and the HP extension library `-lXhp11` must also be linked into the application.

Upon device initialization the proper driver will be loaded. See the discussion of the `gopen` procedure in the **Device Initialization** section of this chapter for details.

Archive Libraries

The name of the archive sox11 driver is `libddsox11.a`. This driver may be linked into your application using an absolute path name `/usr/lib/libddsox11.a`, an appropriate relative path name, or by using the `-l` option `-lddsox11` with the `LDOPTS` environmental variable set to `-a archive`. Xlib `-lx11` and the HP extension library `-lXhp11` must also be linked into the application. The absolute path name of the driver is `/usr/lib/libddsox11.a`.

The reason for using the `LDOPTS` environmental variable is that the `-l` option will look for a shared library driver first and then look for the archive driver if shared was not found. By exporting the `LDOPTS` variable as specified above, the `-l` option will only look for archive drivers. For more information, refer to the *Programming on HP-UX* manual on linking shared or archive libraries.

Examples

To compile and link a C program for use with the shared library driver, use:

```
cc example.c -lddsox11 -L/usr/lib/X11R5 -lXwindow\
-lsb1 -lsb2 -lXhp11 -lX11 -o example
```

or for FORTRAN, use:

```
F77 example.f -lddsox11 -Wl,-L/usr/lib/X11R5 -lXwindow \
-lsb1 -lsb2 -lXhp11 -lX11 -o example
```

or for Pascal, use:

```
pc example.p -lddsox11 -Wl,-L/usr/lib/X11R5 -lXwindow \
-lsb1 -lsb2 -lXhp11 -lX11 -o example
```

Note that `libsb1.a` and `libsb2.a` must be linked *before* the X11 libraries and that `libXhp11.a` must be linked before `libX11.a`.

Both X11 libraries are necessary for proper functionality. The `libX11.a` library allows you to make calls to X, while the `libXhp11.a` library adds HP's extended calls.

Programmatic Initialization

In order to use Starbase functionality within an X11 window, you must perform a `gopen()` call on an existing targeted X11 window. The window may be of any size and location on a device controlled by an X11 server. Multiple processes may `gopen()` the same window (as long as only one process `gopen()` a window as an INDEV), and a single process may `gopen()` multiple windows.

An X11 window for use with the sox11 driver is easily created in either of two ways:

- From outside a program by executing `xwcreate` from a terminal window command line. The use of `xwcreate` is documented in the *Using the X Window System* manual.
- From within a program by calling `XCreateWindow()` and using `make_X11_gopen_string()` to create a string to pass to `gopen()`. The use of `XCreateWindow()` is documented in the *Programming with Xlib* manual. The use of `make_X11_gopen_string()` is documented in the *Starbase Reference* manual.

Gopen Parameters

The `gopen()` procedure has four parameters: Path, Kind, Driver, and Mode.

Path	The pathname of the window as specified to the `xwcreate` command or the string returned by the `make_X11_gopen_string()` command.
Kind	The I/O characteristics of the device. This parameter may be OUTDEV, INDEV, or OUTINDEV for this driver. If OUTDEV is used, then only the output display routines will be available to the application. If INDEV or OUTINDEV is used, then X11 will present a virtual input device interface where a keyboard is present as a CHOICE device and a three button pointer may be accessed as both a CHOICE device and a LOCATOR device.

Note Due to the nature of the X11 protocol, only one process may open a window as INDEV at a time. If more than one process tries to access a window as INDEV, an error will result.

Driver The character representation of the driver type. This parameter may be `NULL` for linking shared or archive libraries - `gopen` will inquire the device and by default load the accelerated driver (if applicable). For example:

`NULL`	for C
`char(0)`	for FORTRAN77
''	for Pascal

Alternatively, a character string may be used to specify a driver. In this case the `UNACCELERATED/ACCELERATED` flag is ignored. For example:

`"sox11"`	for C
`'sox11'//char(0)`	for FORTRAN77
`'sox11'`	for Pascal

Mode The mode control word, which consists of several flag bits which are OR'd together. The RESET_DEVICE and INIT flags clear the window and cause the default Starbase colormap to be set for the sox11 window, but do **NOT** cause any hardware to be reset in the devices.

Gopen Examples

An X11 window must first exist before trying to use `gopen()` with X11. The `xwcreate` command creates a window and a pty file. The file can be used as a device file by `gopen()` to access the window. The name of the file is the name of the window supplied to `xwcreate` prefixed by the file path. The default path is `/dev/screen`. See the `xwcreate` manual page for details.

Three methods for a C program to create and `gopen()` an X11 window follow. These examples create a 150x150 window named `window1` at location `5,5` on the default display. They then `gopen()` `window1` for an output application. Method

#2 is identical to method #1 except that the `xwcreate` command has been moved inside the program. Method #1:

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

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

       int fildes;           /* Starbase graphics descriptor */

       fildes = gopen(argv [1], OUTDEV, argv [2], INIT);
       if(filedes == -1) exit(1);
       ellipse(fildes, 0.3, 0.4, 0.5, 0.5, 0.7);
       make_picture_current(fildes);
       sleep(10);
       gclose(fildes);
}
```

If the above program is called "myprog," then it would be run using the following commands:

```
xwcreate -geometry 150x150+5+5 window1
myprog /dev/screen/window1
xwdestroy window1
```

Method #2:

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

main()
{
     int fildes;           /* Starbase graphics descriptor */

     system("xwcreate -geometry 150x150+5+5 window1");
     fildes = gopen("/dev/screen/window1", OUTDEV, "sox11", INIT);
     if(filedes == -1) exit(1);
     ellipse(fildes, 0.3, 0.4, 0.5, 0.5, 0.7);
     make_picture_current(fildes);
     sleep(10);
     gclose(fildes);
}
```

Method #3:

```
#include <starbase.c.h>
#include <X11/Xlib.h>
```

```
#include <stdio.h>

main()
{
     Display *Xdisplay;      /* X display connection */
     Window   window;        /* X window identifier */
     XEvent event;           /* Holds X server events */
     int fildes;             /* Starbase graphics descriptor */
     extern char *make_X11_gopen_string();

     if ((Xdisplay = XOpenDisplay(NULL)) == NULL) {
         fprintf(stderr, "Can't open %s\en", XDisplayName(NULL));
         exit(1);
     }

     window = XCreateSimpleWindow(Xdisplay,  /*Create the window */
         DefaultRootWindow(Xdisplay),
         5, 5, 150, 150, 2,
         WhitePixel(Xdisplay, DefaultScreen(Xdisplay)),
         BlackPixel(Xdisplay, DefaultScreen(Xdisplay)));

     XSelectInput(Xdisplay, window, StructureNotifyMask);

     XMapWindow(Xdisplay, window);
     XSync(Xdisplay, 0);

     do {            /* Make sure window is visible */
         XNextEvent(Xdisplay, &event);   /* Before writing to it */
     } while (event.type != MapNotify || event.xmap.window != window);

     fildes = gopen(make_X11_gopen_string(Xdisplay, window),  /* Gopen window */
         OUTDEV, "sox11", INIT);
     ellipse(fildes, 0.3, 0.4, 0.5, 0.5, 0.7);       /* Render a picture */
     make_picture_current(fildes);
     sleep(10);
     gclose(fildes);
     XCloseDisplay(Xdisplay);
}
```

Special Device Characteristics

For device coordinate operations, location (0,0) is the upper left corner of the window at the time `gopen()` is executed. Values along the horizontal axis increase to the right. Values on the vertical axis increase in a downward direction.

Device Defaults

The device defaults depend upon the defaults for the display device on which the window is located. These defaults are determined by the X11 server for that display device. For example, an X11 server on a monochrome display defaults to only having one plane of color, black and white, while an X11 server on a pseudo-color display utilizes as many planes of color as the display allows.

Some virtual device defaults are determined by the software driver, `libddsox11.a`. These are as documented below.

Line Type Defaults

Default line types are shown in the table below:

Table 17-1. Default Line Types

Line Type	Bit Pattern
0	1111111111111111
1	1111111100000000
2	1010101010101010
3	1111111111111010
4	1111111111101010
5	1111111111100000
6	1111111111110110
7	1111111110110110

Color Map Defaults

When executing a `gopen` in an X11 window without an INIT flag set, the X11 colormap for the window is read into the Starbase color map. A call to `inquire_color_table` provides information about the color map. The X11 server and direct calls to the X11 library arbitrate over the allocation of color cells.

`define_color_table` is supported. The default color table is set up if a device is opened with the INIT flag set. Note that *the color scheme of all the windows will change* when this call is made and the window receives "colormap focus" (see below).

`define_color_table` will define a virtual colormap for the window. Anytime the "colomap focus" is given to the window by the window manager, that window's virtual colormap will be installed in the hardware colormap . In the window manager (vuewm or mwm), colormap focus can be set in three ways:

pointer	Anytime the pointer enters the window, that window is given the colormap focus.
explicit	Anytime the window receives a button click, it is given colormap focus.
keyboard	Any window that has input focus has colormap focus.

Be aware that the same color may have different values in different colormaps and that switching colormaps affects every window on your screen. For example, if you want to run Starbase on X11, you could run into the following situations.

- If you use the X11 colormap, your X environment has the proper colors, but the Starbase window is strangely colored.
- If you use the Starbase colormap, the Starbase window has the proper colors, but your X environment is strangely colored.

Input Defaults

The keyboard input is by default set to "cooked" mode. This mode returns National Language Support (NLS) values for the full range of ASCII representable keys. Other special function keys return keycodes defined by the reference page for `XrInitMap(3X)`.

Also, by default, only key presses and button presses are reported.

An application can request to be sent raw keystrokes or key and button releases through gescapes. See the section describing the input gescapes for details.

The X11 pointer is CHOICE device ordinal one, a mask of all buttons pressed is CHOICE device ordinal two, the X11 keyboard is CHOICE device ordinal three, and the X11 pointer position is LOCATOR device ordinal one.

Starbase Functionality

Commands Not Supported (NO-OPS)

The following commands are not supported. These commands will not generate Starbase errors.

```
await_retrace                depth_cue_range
backface_control             hidden_surface
bank_switch                  light_ambient
bf_control                   light_attenuation
bf_fill_color                interior_style (INT_OUTLINE, INT_POINT)
bf_interior_style            light_model
bf_perimeter_color           light_source
bf_perimeter_type            light_switch
bf_perimeter_repeat_length   shade_mode
bf_surface_coeficients       shade_range
bf_surface_model             surface_coefficient
dbuffer_switch               surface_model
define_trimming_curve        viewpoint
depth_cue                    zbuffer_switch
depth_cue_color
```

Gescapes

The following gescapes are supported by the sox11 driver. Detailed information on these gescapes can be found in Appendix A of this manual.

- READ_COLOR_MAP
- R_BIT_MASK
- R_BIT_MODE
- R_DEF_FILL_PAT
- TRIGGER_ON_RELEASE
- IGNORE_RELEASE

All other gescapes are used to control the input model. These gescapes are described in the following section (Input Model).

- XN_INPUT_RAW
- XN_KEY_RELEASE
- XN_BUTTON_RELEASE

Input Model

A Starbase application is free to `gopen` an X11 window with the OUTDEV flag and utilize the X11 library calls to perform input. The Starbase application may also choose to use Starbase library input routines if the INDEV or OUTINDEV flags are used as arguments to `gopen`. A program should use exclusively either X11 input routines or Starbase input routines. Using both within the same application may cause an XError.

The sox11 input model represents X11 as a virtual device including a keyboard and an X11 pointer. The keyboard is accessed as a CHOICE device while the X11 pointer is accessed as a CHOICE and a LOCATOR device.

The default mode returns HP Roman-8 keycodes. This is the "cooked" mode for input keystrokes. Special function keys and control keys are not supported in this mode and their return value is undefined. These keys generate a two-byte sequence. The `sample_choice` and `request_choice` functions will return one of the two bytes. The other byte is discarded. (As to which byte is retained, this is undefined.)

"Raw" input for this driver consists of an integer composed of two parts. The first half (or upper two bytes) specify the state of the keyboard at the time of the button or key press (that is, an "Extend char" or "CTRL" key is depressed at the time of the event). The lower half (or last two bytes) is a server dependent key symbol which, for this server, identifies each key or button.

In "cooked" mode the X11 pointer buttons are represented by values one through five, corresponding to the X11 pointer button used. Button one typically represents the left-hand button, two represents the middle button, etc. In "raw" mode the value returned by a button is determined by the "raw" value returned by the X11 server for that button.

The X Window System input device can also return raw keycodes. These codes are the unmapped keycodes and the keyboard state information returned by the X11 server. The input model can be placed in "raw" mode using the XN_INPUT_RAW gescape with a value of TRUE in the first argument of the gescape. Similarly the input model can be reset to the default "cooked" mode using the XN_INPUT_RAW gescape with the first argument containing the value of FALSE.

By default the sox11 driver returns only events associated with the key press and button press input. An application can request input events associated with both presses and releases of keys or buttons. Requesting the release events associated with X11 pointer buttons or keys can be controlled independently.

An application can request events associated with both button presses and button releases by using the XN_BUTTON_RELEASE gescapes. The value returned by a button release event is the negative value of the corresponding button press event.

An application can request to be sent events associated with both key presses and key releases using the XN_KEY_RELEASE gescape. The value returned by a key release event is the negative value of the corresponding key press event.

Both of these gescapes are set by sending a first argument of TRUE. Both gescapes are reset by sending a first argument of FALSE.

HP-HIL pointing devices controlled by the X11 server are mapped into the virtual X11 pointer device represented by the input model. The X11 pointer cannot be controlled by both the X11 server, sox11, and the HP-HIL device driver, `libddhil.a`, at the same time. If the pointer device is to be controlled by the Starbase HP-HIL device driver, then the pointer must be excluded from the X11 server.

Programming Strategy

Directly Calling the X11 Library

A window resize event will have no effect on the space that the driver runs in. At the time of the `gopen()`, the driver determines window size and it will run in that space until `gopen()` is executed in the window again.

Starbase Echo

When a Starbase echo is used, it is removed before every *Series* of Starbase primitives and placed back in the window after drawing is finished. This increases performance dramatically compared to putting the echo back after every primitive, but it means that, after every series of draws, `make_picture_current()` should be called so that the echo will reappear in the window.

X Cursor

When a window is created within a Starbase application by making a call to `XCreateWindow()` or `XCreateSimpleWindow()`, no default X cursor is defined for the window. Instead, the window inherits its cursor from its parent.

If a window is created via the `xwcreate` command, however, the white left arrow cursor is installed by sox11 at `gopen` time.

This allows maximum flexibility for applications creating their own window. Any cursor may be defined for the window using any of the Xlib cursor calls, and that cursor won't be changed by sox11.

Polygon Fills

The `sox11` driver uses the following polygon fill algorithm: A border pixel is drawn only if the polygon is to the right of or underneath the border pixel. This allows two polygons using the exclusive OR rule to be drawn next to each other without any loss of continuity.

Window Mapping

When running in a “window smart” environment, be sure to map the created window and do an `XSync` before gopening sox11. This ensures that your window exists, and that no drawing calls will be lost. In order to ensure all drawing calls are displayed, you must create a retained window or trap all exposure events and redraw the portion that was previously occluded or unmapped.

Double Buffering

Double buffering is accomplished by modifying the colormaps for the Starbase window. See the “Colormap Defaults” section for details. Be aware, however, of the problems that can occur when a virtual colormap is changed.

Note that double buffering is not supported in CMAP_FULL mode.

Raster Text

If you wish to get the most efficient performance from calls to `fm_write`, set the colormode flag to FALSE. See the fast alpha/font manager documentation for details.

The following fast alpha/font manager calls are not supported when rendering to a remote window:

```
fm_kjfontinfo
fm_rasterfontinfo
```

Raster text to non-HP equipment is not supported.

The HP-HIL Device Driver

Device Description

The Hewlett-Packard Human Interface Link (HP-HIL) Device Driver is used to provide graphics input from the following devices:

- HP 45911A HP-HIL Graphics Tablet
- HP 46020A HP-HIL Keyboard
- HP 46021A HP-HIL Keyboard
- HP 46060A HP-HIL Mouse
- HP 46060B HP-HIL 3-Button Mouse
- HP 46083A HP-HIL Knob
- HP 46085A HP-HIL Control Dial Module
- HP 46086A HP-HIL 32-Button Box
- HP 46087A HP-HIL A-Size Digitizer
- HP 46088A HP-HIL B-Size Digitizer
- HP 46089A HP-HIL 4-Button Cursor for the HP 46087/88A Tablets
- HP 46094A HP-HIL Quadrature Box
- HP 46095A HP-HIL Quadrature 3-Button Mouse
- HP 80409A HP-HIL 3-Button Track Ball

Setting Up the Device

Special Device Files (mknod)

The `mknod` command creates a special device file which is used to communicate between the computer and the peripheral device. See the `mknod`(1M) information in the *HP-UX Reference* for further information. The name of this special device file is passed to Starbase in the `gopen` procedure. Since superuser capabilities are needed to create special device files, they are normally created by the system administrator.

Although special device files can be made in any directory of the HP-UX file system, the convention is to create them in the `/dev` directory. Any name may be used for the special device file, however the name that is suggested for these devices is `hil1` for the first device on the hil loop, `hil2` for the next device, etc.

There may be up to seven devices connected to a single HP-HIL driver board allowing device file names of the form `hil1`, `hil2`, ... , `hil7`.

The HP 46085A HP-HIL Control Dial Module must have three device files created for it since each set of three dials in a row acts as a HP-HIL device.

The following examples will create a special device file for this device. Remember that you must be superuser or root to use the `mknod` command.

For the Series 300

The `mknod` parameters should create a character device with a major number of 24 and a minor number of `0x0000`⟨*a*⟩ where ⟨*a*⟩ is the device's one digit address (position on the HP-HIL loop from the computer interface card).

```
mknod /dev/hil⟨a⟩ c 24 0x0000⟨a⟩0
```

For the Series 700

The `mknod` parameters should create a character device with a major number of 24 and a minor number of `0x2030`⟨*a*⟩`0` where⟨*a*⟩ is the device's one digit address (position on the HP-HIL loop from the computer interface card).

```
   mknod /dev/hil⟨a⟩ c 24 0x2030⟨a⟩0
```

For the Series 800

The `mknod` parameters should create a character device with a major number of 24 and a minor number of `0x00`⟨*lu*⟩⟨*a*⟩`0` where ⟨*lu*⟩ is the two-digit hardware logical unit and ⟨*a*⟩ is the device's one-digit address (position from the computer interface card). Note that the `0x` causes the number to be interpreted hexadecimally.

```
   mknod /dev/hil⟨a⟩ c 24 0x00⟨lu⟩⟨a⟩0
```

or

```
   mknod /dev/hil_⟨lu⟩.⟨a⟩ c 24  0x00⟨lu⟩⟨a⟩0
```

Linking the Driver

Shared Libraries

The shared HP HP-HIL Device Driver is the file named `libddhil.sl` in the `/usr/lib` directory. The device driver will be explicitly loaded at run time by compiling and linking with the starbase shared library `/usr/lib/libsb.sl`, or by using the `-l` option `-lsb`.

Examples

To compile and link a C program for use with the shared library driver, use:

```
   cc example.c -I/usr/include/X11R5/x11 -L/usr/lib/X11R5\
   -lXwindow -lsb -lXhp11 -lX11 -ldld -lm -o example
```

or with FORTRAN use,

```
   F77 example.f -Wl,-L/usr/lib/X11R5 -lXwindow -lsb\
   -lXhp11 -lX11 -ldld -lm  -o example
```

or with Pascal use,

```
pc example.p  -Wl,-L/usr/lib/X11R5 -lXwindow -lsb\
-lXhp11 -lX11 -ldld -lm -o example
```

For details, see the discussion of the `gopen` procedure in the section *To Open and Initialize the Device* in this chapter.

Archive Libraries

The archive HP-HIL Device Driver is located in the `/usr/lib` directory with the file name `libddhil.a` . This device driver may be linked to a program by using the absolute path name `/usr/lib/libddhil.a`, an appropriate relative path name, or by using the `-l` option as in `-lddhil`.

The `LDOPTS` environmental variable must be set to `-a archive`. The reason for using this environmental variable is that the `-l` option will look for a shared library driver first and then look for the archive driver if shared was not found. By exporting the `LDOPTS` variable as specified above, the `-l` option will only look for archive drivers. For more information, refer to the *Programming on HP-UX* manual on linking shared or archive libraries.

Examples

Assuming you are using `ksh`(1), to compile and link a C program for use with this driver, use:

```
export LDOPTS="-a archive"
```

and then:

```
cc example.c -lddhil -L/usr/lib/X11R5 -lXwindow\
-lsb1 -lsb2 -lXhp11 -lX11 -lm  -o example
```

or for FORTRAN, use:

```
F77 example.f -lddhil -Wl,-L/usr/lib/X11R5 -lXwindow\
 -lsb1 -lsb2 -lXhp11 -lX11 -o example
```

or for Pascal, use:

```
pc example.p -lddhil -Wl,-L/usr/lib/X11R5 -lXwindow\
 -lsb1 -lsb2 -lXhp11 -lX11 -o example
```

Device Initialization

Parameters for gopen

The `gopen` procedure has four parameters: Path, Kind, Driver, and Mode.

Path	When opening a device for exclusive access, this parameter is the name of the special device file created by the `mknod` command as specified in the last section, that is, `/dev/hil1`. When opening a device for shared access in an X11 environment, this parameter describes a `device/window combination`. Please refer to the chapter "Input Operation" in the *Starbase Graphics Techniques* manual.
Kind	Indicates the I/O characteristics of the device. This parameter must be `INDEV` for this driver.
Driver	The character representation of the driver type. This is `hp-hil` modified to meet the syntax of the programming language used, namely:
Mode	This parameter is ignored.

`"hp-hil"`	*for C.*
`'hp-hil'//char(0)`	*for FORTRAN77.*
`'hp-hil'`	*for Pascal.*

Syntax Examples

To open and initialize an HP-HIL Mouse device at the second position on the loop for input:

For C Programs:

```
fildes = gopen("/dev/hil2",INDEV,"hp-hil",INIT);
```

For FORTRAN77 Programs:

```
fildes = gopen('/dev/hil2'//char(0), INDEV,'hp-hil'//char(0),INIT)
```

For Pascal Programs:

```
fildes = gopen('/dev/hil2',INDEV,'hp-hil',INIT);
```

Special Device Characteristics

At each HP-HIL address there can be:

- 0, 1 or 2 locator devices (each can return X,Y,Z values)
- 0 or 2 choice device types

Enabling events with `class = ALL` will enable all of the above that are present. If a choice device is present, you will get two choice events for each button press. One is the button number, the other is the 32-bit wide bit map.

Some locator devices, such as the HP 46085A HP-HIL Control Dial Module 9-Knob Box, have no buttons on them. This means that they can only be sampled. Request and event functions have no meaning for these devices.

Cautions

1. Opening up a keyboard `hil` device in a non-window environment will take away `ITE` keyboard access from the console until that device is closed.
2. Forking a process while an `hil` device is open will keep that device open until the child process is complete or the device is explicitly closed by the child.

Starbase Functionality

Locator Devices

There are several defaults created at `gopen` time.

Relative Positioning

For some HP-HIL locator devices, position location is relative to the initial position of the locator device (0, 0, 0).

For example, the initial "position" of a mouse is (0, 0, 0). Any move by the mouse is with respect to that position. All relative devices have a default P1, P2 locator area that is 20 centimeters square. Movement of relative devices beyond the P1, P2 limits is ignored and the device remains *located* at the point the device crossed the P1, P2 boundary.

If you move beyond the P1, P2 limits and then move the relative device in the reverse direction, the device position immediately starts to change. All motion beyond the clip limit is forgotten, and the reversal point becomes the new limit position.

If the procedure `set_p1_p2` is executed with the `FRACTIONAL` parameter, the fraction is with respect to 20 centimeters.

If the procedure `set_p1_p2` is executed with the `METRIC` parameter, the limit values are unlimited and can be larger than 20 centimeters.

The initial *position* of relative devices such as a mouse can be set via the `set_locator` procedure.

Movement of the mouse is converted from device units to virtual device coordinate values. The size of a device unit can be found using the `inquire_sizes` function. To change the reference point, use the `set_locator` procedure. All location coordinates are clipped to the rectangle defined by P1 and P2.

The default P1, P2 area for relative `hil` devices is square. To get a mapping from the full range of the input device to the full range of the output device, either call `set_p1_p2` with `METRIC` parameters that have an aspect ratio equal to the aspect ratio of the output device or call `mapping_mode(DISTORT)`.

Absolute Positioning

For some HP-HIL locator devices, position is absolutely defined. Information concerning the limits of these devices is provided with the manuals for these devices.

To find the resolution of the device, use the `inquire_sizes` procedure.

Input values are not clipped, and absolute devices may return points outside of the VDC extent.

Choice Devices

Choice devices are divided into two groups.

- Ordinal 1—Reports the button number as an integer. Pressing a button returns a positive value. By default, releasing a button will return zero. If the `gescape` `TRIGGER_ON_RELEASE` has been executed, releasing a button will return a negative valued button number.
- Ordinal 2—Reports a 32-bit wide bit mask. The least significant bit equals button 1 and the most significant bit equals button 32. A one value indicates that the button is pressed. Buttons greater than 32 will trigger this report, but will not affect the bit mask returned. Releasing a button will cause the corresponding bit to be reset to zero.

HP-HIL Keyboards

If an HP-HIL keyboard is accessed using the Starbase `gopen` call, the keyboard will no longer report to the terminal emulator. Be sure to leave a way to `gclose` since the break key will not stop the program.

All keyboards are considered to be USASCII keyboards. When a key is depressed, the USASCII integer value of that key is returned. Exceptions are $\boxed{\text{f1}}$ thru $\boxed{\text{f8}}$ plus the four unmarked keys in the upper-right corner of the keyboard representing keys 1 thru 12 respectively.

Table 18-1. Keys and Their Values

Key	Value	Key	Value	Key	Value
Break	232	Delete line	240	System	248
Reset	233	Clear line	241	User	249
Stop	234	Clear display	242	Prev	250
Stop	234	Clear display	242	Prev	250
Extend char (left)	235	Menu	243	Next	251
Extend char (right	236	HOME	244	up arrow	252
Insert char	237	Select	245	down arrow	253
Delete char	238	Enter	246	right arrow	254
Insert line	239	Print	247	left arrow	255

Devices Without Triggers

Some devices provide location data, but have no buttons. Since they have no trigger action, special rules apply to them.

- All requests are invalid.
- `inquire_request_status` is *never* `TRUE` (1). This means use sample procedures only.
- There is no way to generate an event.

Parameters for gescape

The `hp-hil` driver supports the following `gescapes`. Detailed information about `gescape` functions can be found in Appendix A.

- `IGNORE_RELEASE`—Trigger when button pressed.
- `TRIGGER_ON_RELEASE`—Trigger when button released.

These `gescape` functions are unique to this driver:

- `DISABLE_AUTO_PROMPT`—Disable HP-HIL auto prompt.
- `ENABLE_AUTO_PROMPT`—Enable HP-HIL auto prompt.
- `IGNORE_PROXIMITY`—Ignores stylus proximity.
- `PROMPT_OFF`—Switch prompt indicator off.
- `PROMPT_ON`—Switch prompt indicator on.
- `REPORT_PROXIMITY`—Reports stylus proximity.
- `SET_ACCELERATION`—Set acceleration and threshold values.

19

The HP Keyboard Device Driver

Device Description

This driver allows an Hewlett-Packard keyboard to be used as a choice device. When an event occurs, the ordinal ASCII value of the key depressed is returned.

Setting Up the Device

Special Device Files

The device file used for the keyboard that you are logged into is in the `/dev` directory with the file name `tty`. For other keyboards on your system, your system administrator may set up different device files. See your system administrator for information about those files.

Linking the Driver

Shared Libraries

The shared keyboard device driver is the file named `libddkbd.sl` in the `/usr/lib` directory. The device driver will be explicitly loaded at run time by compiling and linking with the starbase shared library `/usr/lib/libsb.sl`, or by using the `-l` option `-lsb`.

Examples

To compile and link a C program for use with the shared library driver, use:

```
cc example.c -I/usr/include/X11R5/x11 -L/usr/lib/X11R5\
-lXwindow -lsb -lXhp11 -lX11 -ldld -lm -o example
```

or with FORTRAN use,

```
F77 example.f -Wl,-L/usr/lib/X11R5 -lXwindow -lsb\
-lXhp11 -lX11 -ldld -lm  -o example
```

or with Pascal use,

```
pc example.p  -Wl,-L/usr/lib/X11R5 -lXwindow -lsb\
-lXhp11 -lX11 -ldld -lm -o example
```

For details, see the discussion of the `gopen` procedure in the section *To Open and Initialize the Device* in this chapter.

Archive Libraries

The archive keyboard device driver is located in the `/usr/lib` directory with the file name `libddkbd.a`. This device driver may be linked to a program by using the absolute path name `/usr/lib/libddkbd.a`, an appropriate relative path name, or by using the `-l` option as in `-lddkbd` with the `LDOPTS` environmental variable set to `-a archive`.

The reason for using this environmental variable is that the `-l` option will look for a shared library driver first and then look for the archive driver if shared was not found. By exporting the `LDOPTS` variable as specified above, the `-l` option will only look for archive drivers. For more information, refer to the *Programming on HP-UX* manual on linking shared or archive libraries.

Examples

Assuming you are using ksh(1), to compile and link a C program for use with this driver, use:

```
export LDOPTS="-a archive"
```

and then:

```
cc example.c -lddkbd -L/usr/lib/X11R5 -lXwindow\
-lsb1 -lsb2 -lXhp11 -lX11 -lm  -o example
```

or for FORTRAN, use:

```
F77 example.f -lddkbd -Wl,-L/usr/lib/X11R5 -lXwindow\
 -lsb1 -lsb2 -lXhp11 -lX11 -o example
```

or for Pascal, use:

```
pc example.p -lddkbd -Wl,-L/usr/lib/X11R5 -lXwindow\
 -lsb1 -lsb2 -lXhp11 -lX11 -o example
```

Device Initialization

Parameters for gopen

The `gopen` procedure has four parameters: Path, Kind, Driver, and Mode.

Path	This is the name of the special device file created by the `mknod` command as specified in the last section. For example, `/dev/tty`.
Kind	This indicates the `I/O` characteristics of the device. The parameter must be `INDEV` for this driver.
Driver	This is the character representation of the driver type. For this driver, use `keyboard` or `kbd` modified to meet the syntax of the programming language used. For example, use one of the following appropriate for the language being used:

`"keyboard"`	*for C.*
`"kbd"`	*for C.*

`'keyboard'//char(0)`	*for FORTRAN77.*
`'kbd'//char(0)`	*for FORTRAN77.*
`'keyboard'`	*for Pascal.*
`'kbd'`	*for Pascal.*

Mode This parameter is ignored.

Syntax Examples

To open and initialize a keyboard device for input:

For C Programs:

```
fildes = gopen("/dev/tty",INDEV,"kbd",INIT);
```

For FORTRAN77 Programs:

```
fildes = gopen('/dev/tty'//char(0),INDEV,'kbd'//char(0),INIT)
```

For Pascal Programs:

```
fildes = gopen('/dev/tty',INDEV,'kbd',INIT);
```

Special Device Characteristics

- The keyboard driver only supports one choice subdevice.
- No locator functions are currently supported.
- To get access to local keys (such as arrow keys), the transmit function's escape code should be sent to the `tty` before accessing the `tty` with the `gopen` command. These functions are $^{E}{}_{C}$`&s1A` to transmit local functions, and $^{E}{}_{C}$`&s0A` not to transmit local functions.
- The HP-HIL driver can also be used to access the HIL keyboard, but only one driver (HIL or keyboard) can access the keyboard with a `gopen` command at any one time.
- The keyboard driver and the terminal driver cannot be used simultaneously for input from the same device because they interfere with each other's operation.

Starbase Functionality

Since `tty` devices do not generate key transitions (key up and key down), `sample_choice` command gives it best approximation. When events are enabled, the choice value returned (if any) is the last key pressed in the last one half second. If events are not enabled, the choice value returned (if any) is the last key pressed since the last `sample_choice` command or choice request.

At `gopen` time, the keyboard driver performs several tasks that should be noted. It saves and replaces any signal handlers with its own handlers (except for the `SIGKILL`, non-terminating, and ignored signals). Then the current state of the `tty` (see `tty`(4)) is inquired and saved. The state of the `tty` is then changed (to Canonical Input, No Echo, One character blocking reads, etc.) using `ioctl` and `fcntl`. If a signal is received by the current process, one of the keyboard signal handlers is called. This signal handler restores the old state of the `tty` and then calls the signal handler that was present at `gopen` time.

If events are enabled and the current process gets killed by any signal, the Starbase daemon program will also restore the state of the `tty`. This is done in case a `SIGKILL` was received. If events are not enabled and the current process gets killed, the `tty` is left in a bad state. To fix this try typing:

```
CONTROL J   stty hp   CONTROL J
```

Sophisticated users that need to use their own signal handlers and/or change the state of the `tty` should be aware of these operations and program around them.

The keyboard driver uses the `sigvector` system call to set up its signal handlers. The `sigvector` system call is incompatible with the `signal` system call. Thus, users who need to use their own signal handlers will need to use `sigvector`. For more information, see `sigvector(2)` and `signal(2)` in the *HP-UX Reference*.

The HP Locator Keyboard Device Driver

Device Description

This driver allows a Hewlett-Packard keyboard to be used as a choice input device. Arrow keys can be used as a locator device if they are described in the `terminfo`(4) data base.

The keyboard is treated as an ASCII device and is accessed via the `termio`(7) interface. The HP-HIL Device Driver provides `raw` access to HIL keyboards.

Setting Up the Device

Special Device Files (mknod)

The `mknod` command creates a special device file which is used to communicate between the computer and the peripheral device. Refer to the `mknod`(1M) entry in the *HP-UX Reference* for further information. The name of this special device file is passed to Starbase in the `gopen` procedure. Since superuser capabilities are needed to create special device files, they are normally created by the system administrator.

Although special device files can be made in any directory of the HP-UX file system, the convention is to create them in the `/dev` directory. The special device file `/dev/tty` always refers to the keyboard at which you are logged in to the system. For other keyboards on your system, your system administrator may set up different device files. Normally, for a terminal keyboard, the device file used to access the terminal can be used, but a `getty`(1M) process also running on the terminal will interfere with correct behavior of the keyboard device. Logging in to the other terminal and executing a long `sleep`(1) is one way to temporarily disable the `getty` process; the `sleep` can normally be terminated with the BREAK key. For the same reason, a program cannot concurrently get both user textual input and graphics input from the same keyboard.

The following example will create a `/dev/tty` special device file. Remember that you must be superuser to use the `mknod` command. This file usually exists and therefore does not need to be created.

```
mknod /dev/tty c 2 0x000000 RETURN
```

Note that the leading `0x` causes the number be interpreted hexadecimally.

Linking the Driver

Shared Libraries

20 The shared locator keyboard device driver is the file named `libddlkbd.sl` in the `/usr/lib` directory. The device driver will be explicitly loaded at run time by compiling and linking with the starbase shared library `/usr/lib/libsb.sl`, or by using the `-l` option `-lsb`.

Examples

To compile and link a C program for use with the shared library driver, use:

```
cc example.c -I/usr/include/X11R5/x11 -L/usr/lib/X11R5\
-lXwindow -lsb -lXhp11 -lX11 -ldld -lm -o example
```

or with FORTRAN use,

```
F77 example.f -Wl,-L/usr/lib/X11R5 -lXwindow -lsb\
-lXhp11 -lX11 -ldld -lm  -o example
```

or with Pascal use,

```
pc example.p  -Wl,-L/usr/lib/X11R5 -lXwindow -lsb\
-lXhp11 -lX11 -ldld -lm -o example
```

For details, see the discussion of the `gopen` procedure in the section *To Open and Initialize the Device* in this chapter.

Archive Libraries

The archive locator keyboard device driver is located in the `/usr/lib` directory with the file name `libddlkbd.a`. This device driver may be linked to a program by using the `-l` option as in `-lddlkbd` with the `LDOPTS` environmental variable set to `-a archive`. The driver also requires the `curses`(3) library to be linked.

The reason for using the `LDOPTS` environmental variable is that the `-l` option will look for a shared library driver first and then look for the archive driver if shared was not found. By exporting the `LDOPTS` variable as specified above, the `-l` option will only look for archive drivers. For more information, refer to the *Programming on HP-UX* manual on linking shared or archive libraries.

Examples

Assuming you are using `ksh`(1), to compile and link a C program for use with this driver, use:

```
export LDOPTS="-a archive"
```

and then:

```
cc example.c -lddlkdb -L/usr/lib/X11R5 -lXwindow\
-lsb1 -lsb2 -lXhp11 -lX11 -lm  -o example
```

or for FORTRAN, use:

```
F77 example.f -lddlkdb -Wl,-L/usr/lib/X11R5 -lXwindow\
 -lsb1 -lsb2 -lXhp11 -lX11 -o example
```

or for Pascal, use:

```
pc example.p -lddlkbd -Wl,-L/usr/lib/X11R5 -lXwindow\
 -lsb1 -lsb2 -lXhp11 -lX11 -o example
```

Terminfo Support Required

The locator keyboard driver uses the `terminfo`(5) data base and `curses`(3) to enable and recognize the escape sequences sent by the terminal arrow keys. In order to do this, the `terminfo` data base entry for your current terminal (as indicated by the `TERM` environment variable) must include the necessary items. If these items are not present, the choice device will still function, but the locator device will not work properly.

Modifying a `terminfo` entry should be done in several steps:

1. The current entry can be placed in a text file for editing by using `untic`, which reverses the effect of the `tic`(1M) processing program:

   ```
   untic $TERM >myentry
   ```

2. The following items must be added to the entry if not present:

Table 20-1.

Capability Description	Capability ID
enable keypad	smkx
disable keypad	rmkx
up arrow.	kcuu1
down arrow	kcud1
right arrow	kcuf1
left arrow	kcub1
scroll up	kind
scroll down	kri
home up	khome
home down	kll

For example, to extend the standard Hewlett-Packard terminal, the entry must include the following items (the first six of which are usually included in the `terminfo` entry as shipped with HP-UX):

```
smkx=\E&s1A
rmkx=\E&s0A
kcuu1=\EA
kcud1=\EB
kcuf1=\EC
kcub1=\ED
kind=\ES
kri=\ET
khome=\Eh
kll=\EF
```

3. You should set your `TERMINFO` environment variable to a local directory for testing purposes. The system will look in this directory first when attempting to set up a terminal interface.

   For the C shell:

   ```
   setenv TERMINFO /users/joe/term
   ```

   For the Bourne shell:

   ```
   TERMINFO=/users/joe/term
   export TERMINFO
   ```

4. The `tic`(1M) processor is used to compile the modified entry. The `tic` will use the current value of `TERMINFO` as the base directory for its output. Be forewarned: `tic` creates subdirectories as necessary in the base directory.

   ```
   tic -v myentry
   ```

5. Finally, the entry should be tested to make sure it is correct. When that has been determined, the entry can be recompiled (by the superuser) into the system default base directory, `/usr/lib/terminfo`. This should not be done until it is absolutely certain that the entry is correct and that no information has been lost from the original. After the entry is placed in the default location, the `TERMINFO` environment variable need no longer be set to gain access to the modified entry.

Initialization

Parameters for gopen

The `gopen` procedure has four parameters: Path, Kind, Driver, and Mode.

Path	The name of the special device file created by the `mknod` command as specified in the last section, e.g. `/dev/tty`.
Kind	Indicates the `I/O` characteristics of the device. This parameter must be `INDEV` for this driver.
Driver	The character representation of the driver type. This is `lkbd` modified to meet the syntax of the programming language used, namely:

`"lkbd"`	*for C.*
`'lkbd'//char(0)`	*for Fortran77.*
`'lkbd'`	*for Pascal.*

Mode	The mode control word (consists of several flag bits *or* ed together. For this driver, the mode parameter is ignored. The driver always starts with default values for locator position and resolution as described in the following examples.

Syntax Examples

To open and initialize a keyboard device for output:

For C programs:

```
fildes = gopen("/dev/tty",INDEV,"lkbd",INIT);
```

For FORTRAN77 programs:

```
fildes = gopen('/dev/tty'//char(0), INDEV,'lkbd'//char(0),INIT)
```

For Pascal programs:

```
fildes = gopen('/dev/tty',INDEV,'lkbd',INIT);
```

Special Device Characteristics

The locator keyboard driver replaces handlers for most signals with its own cleanup routine in order to restore your keyboard processing to its state before `gopen` was called. If you have specified handlers for any signals, they will be called after the cleanup. Cleanup is not done for `SIGPWR`, `SIGKILL`, `SIGCLD`, or `SGWINDOW`. Your handler is restored at `gclose`.

Input processing is set to canonical, no echo, one-character non-blocking reads while the driver is opened. Should the driver be killed in such a way that it cannot clean up, the `tty` may be left in a bad state. To fix this, try typing:

```
CONTROL J stty hp CONTROL J
```

Sophisticated users that need to use their own signal handlers and/or change the state of the `tty` should be aware of the locator keyboard driver behavior and program their applications accordingly.

The locator keyboard driver uses the `sigvector` system call to set up its signal handlers. The `sigvector` system call is incompatible with the `signal` system call. Thus, users who need to use their own signal handlers will need to use `sigvector`. For more information, see `sigvector(2)` and `signal(2)` in the *HP-UX Reference*.

Starbase Functionality

Choice Devices

The driver supports one choice device. When used as a choice device, the ordinal ASCII value of the key depressed is returned. Since key transitions cannot be detected, `sample_choice` will return the key pressed most recently in the last 0.1 second. If a tenth of a second has elapsed since a keypress, the choice value returned will be zero. An exception to this rule is the ASCII (escape) key. The `curses` routines, in attempting to recognize escape sequences sent by the keypad, will wait one second before deciding that the (escape) key (value 27) has in fact been pressed. Escape sequences that are not recognized (due to their absence from the `terminfo` data base entry) will be interpreted as two or more keypresses, the first is (escape).

Locator Devices

When the locator is enabled, the alphanumeric keypad arrow keys change the locator position one unit in the appropriate direction. Certain other keys have been defined to change the locator position by ten units rather than one. The supported set of locator keys is:

`up arrow`	`increment y by one unit`
`down arrow`	`decrement y by one unit`
`right arrow`	`increment x by one unit`
`left arrow`	`decrement x by one unit`
`scroll up`	`increment y by ten units`
`scroll down`	`decrement y by ten units`
`home up`	`increment x by ten units`
`home down`	`decrement x by ten units`

These functions do not map to the same keys on all keyboards. Some keyboards may not support the second set of four keys. This will not prevent the arrow keys from functioning properly as long as the `terminfo` entry describes them.

On the ITF keyboard normally used with Series 300 systems, the four `fast` locator keys are mapped as follows:

```
fast up     shift + up arrow
fast down   shift + down arrow
fast right  shift + home
fast left   home
```

One locator device is supported. The locator position is relative to the initial position of the device; the default is (0, 0, 0). The initial position can be set by the `set_locator` procedure.

The initial resolution of the locator is 1024×1024. The resolution of the locator can be changed by calling `set_p1_p2`. One keystroke corresponds to a motion of one device unit in X or Y, and also is defined as one millimeter for purposes of the `set_p1_p2` call with the `METRIC` parameter. If the `FRACTIONAL` parameter is used, the fractions will be multiplied by the 1024×1024 device extents. If `METRIC` is used, the number of millimeters exactly specifies the number of units in the locator limits. This allows mapping of any desired number of **clicks** to the display being used during tracking.

Movement of the locator beyond the current P1, P2 limits is ignored; the device remains located at the point at which the P1, P2 limit is reached. To get a mapping from the full range of the input device to the full range of the output device, call either `set_p1_p2` with `METRIC` parameters that have an aspect ratio equal to the aspect ratio of the output device, or call `mapping_mode` with the ⟨*distort*⟩ parameter `TRUE`.

The arrow keys provide no natural trigger for locator events and requests. Consequently, any ordinary key is considered a trigger for the locator. This means that if both a choice request and a locator request are pending, both will be satisfied at the time of the choice input. However, the locator request may timeout if no key is pressed. Similarly, locator events are captured at the time of a choice keypress. If both locator and choice events are enabled, the choice keypress will cause two simultaneous events to be queued, one from the locator and one from the choice device.

Limitations

Due to the serial processing used on keyboard inputs and the operation of some keyboards, some combinations of input functions are not possible. For example, simultaneously tracking from the locator device and sampling the choice device does not work well because most keyboards do not operate in a continuous **rollover** mode. In other words, holding down one of the arrow keys and a choice key will not cause a stream of alternating arrow and choice keystrokes to be sent to the host computer. Instead, the last key pressed, or perhaps the first key in the scanning sequence built into the keyboard, will be sent repeatedly. In general, continuous high-speed sampling of a serial device is not advisable.

The `lkbd`, `kbd`, and `hpterm` drivers will interfere with each other if any combination is used simultaneously for input from the same terminal.

Parameters for gescape

The following gescape functions are supported by the `lkbd` driver. Detailed information about these functions can be found in Appendix A of this manual:

- `ENABLE_ACKNOWLEDGE`—Allows bell character when request/event is satisfied.
- `DISABLE_ACKNOWLEDGE`—Disables bell function.

21

The HP-GL Device Driver

Device Description

The Hewlett-Packard Graphics Language (HP-GL) Device Driver is a least-common-denominator HP-GL command-set driver. All standard HP-GL command set devices should work properly with this driver. Hewlett-Packard has tested and supports the following HP-GL devices with HP-IB interfaces and serial (RS-232) interfaces for plotters:

- HP 9111A tablet
- HP 7440A plotter
- HP 7470A plotter
- HP 7475A plotter
- HP 7550A plotter
- HP 7570A plotter
- HP 7575A plotter
- HP 7576A plotter
- HP 7580A plotter
- HP 7580B plotter
- HP 7585B plotter
- HP 7586B plotter
- HP 7595A plotter
- HP 7596A plotter
- HP C1600A plotter
- HP C1601A plotter

Setting Up the Device

Switch Settings

For operation of a device with HP-IB interface, the HP-IB address must be the same as the device file address (see "Special Device Files (mknod)").

For operation of this device with RS-232 the plotter must be configured by the user where applicable as follows:

- 8-bit character size
- No parity
- Desired baud rate
- One stop bit if baud rate is greater than 110, otherwise two bits

The device driver "`libddhpgl.a`" automatically configures the plotter to the following:

- `XON/XOFF` protocol with `dc1` and `dc3` signals
- "`;`" command terminator
- ⟨*newline*⟩ response terminator

The device driver "`libddhpgl.a`" also sets the `termio`(4) structure for the device interface to the following:

- 8-bit character size
- `XON/XOFF` protocol
- No parity
- Disable signals `INTR` and `QUIT`
- 2400 baud rate if initially 300
- No postprocessing
- Canonical processing
- Turn off `ERASE` and `KILL` symbols

Note There must *not* be a `getty` running on the serial device file. The following command will `sleep` a `getty`:

```
sleep 2000000000 < /dev/plts
```

Note

If the device is a SPOOLED file, the termio(4) structure for the device interface will *not* be automatically configured, and the user must configure the interface.

The default values for a newly opened interface are:

`300 cs8 cread hupcl` (see *termio*(4), *stty*(1))

The following commands will correctly configure the device interface that already has the above defaults:

```
sleep 2000000000 < /dev/plts &
stty ⟨baud⟩
   ixon ignbrk icanon isig clocal < /dev/plts
stty erase ^- kill ^- < /dev/plts
```

where ⟨*baud*⟩ is the baud rate of the device (600, 1200, 2400, etc.), and /dev/plts is the device file for the serial plotter.

Special Device Files (mknod)

The mknod command creates a special device file which is used to communicate between the computer and the peripheral device. See the mknod(1M) information in the *HP-UX Reference* manual for further information. The name of this special device file is passed to Starbase in the gopen procedure. Since superuser capabilities are needed to create special device files, they are normally created by the system administrator.

Although special device files can be made in any directory of the HP-UX file system, the convention is to create them in the /dev directory. Any name may be used for the special device file. The following examples will create a special device file for this device. Remember that you must be the superuser (the root user) to use the mknod command.

For the Series 300 and 400

HP-IB Card Device File

The `mknod` parameters should create a character device file with a major number of 21 and a minor number of `0x`⟨*sc*⟩⟨*ad*⟩`00` where ⟨*sc*⟩ is the select code and ⟨*ad*⟩ is the device's address.

```
mknod /dev/hpgl c 21 0x⟨sc⟩⟨ad⟩00
```

Serial Interface Card Device File

The `mknod` parameters should create a character device file with a major number of 1 and a minor number of `0x`⟨*sc*⟩⟨*ad*⟩`04` where ⟨*sc*⟩ is the select code and ⟨*ad*⟩ is the port address.

```
mknod /dev/hpgl c 1 0x⟨sc⟩⟨ad⟩04
```

For the Series 700

Serial RS-232 Interface

For Serial Port A, the `mknod` parameters should create a character device file with a major number of 1 and a minor number of `0x204004`:

```
mknod /dev/plts c 1 0x204004
```

For Serial Port B, the `mknod` parameters should create a character device file with a major number of 1 and a minor number of `0x205004`:

```
mknod /dev/plts c 1 0x205004
```

Centronics Parallel Interface

The `mknod` parameters should create a character device file with a major number of 11 and a minor number of `0x206002`:

```
mknod /dev/plt_parallel c 11 0x206002
```

For the Series 800

HP-IB Card Device File

The `mknod` parameters should create a character device file with a major number of 21 and a minor number of `0x00`⟨*lu*⟩⟨*ad*⟩ where ⟨*lu*⟩ is the hardware logical unit and ⟨*ad*⟩ is the device's address.

```
mknod /dev/hpgl c 21 0x00⟨lu⟩⟨ad⟩
```

Serial Interface Card Device File

The `mknod` parameters should create a character device file with a major number of 1 and a minor number of `0x00`⟨*lu*⟩⟨*ad*⟩ where ⟨*lu*⟩ is the hardware logical unit and ⟨*ad*⟩ is the port address.

```
mknod /dev/hpgl c 1 0x00⟨lu⟩⟨ad⟩
```

Linking the Driver

Shared Libraries

The shared HP-GL Device Driver is the file named `libddhpgl.sl` in the `/usr/lib` directory. The device driver will be explicitly loaded at run time by compiling and linking with the starbase shared library `/usr/lib/libsb.sl`.

Note that use of the library `libdvio.a` requires the use of `-Wl,-E` when using plotter driver shared libraries as shown in the **Examples** section.

Examples

To compile and link a C program for use with the shared library driver, use:

```
cc example.c -L/usr/lib/X11R5 -Wl,-E -lddhpgl\
-lXwindow -lsb -lXhp11 -lX11 -ldvio -ldld -lm -o example
```

or with FORTRAN use,

```
F77 example.f -Wl,-L/usr/lib/X11R5 -Wl,-E -lddhpgl\
-lXwindow -lsb -lXhp11 -lX11 -ldvio -ldld -o example
```

or with Pascal use,

```
pc example.p -Wl,-L/usr/lib/X11R5 -Wl,-E -lddhpgl\
-lXwindow -lsb -lXhp11 -lX11 -ldvio -ldld -o example
```

For details, see the discussion of the `gopen` procedure in the section *To Open and Initialize the Device* in this chapter.

Archive Libraries

The archive HP-GL Device Driver has a file name of `libddhpgl.a` and is located in the `/usr/lib` directory. This device driver may be linked to a program by using the absolute path name `/usr/lib/libddhpgl.a`, an appropriate relative path name, or by using the `-l` option as in `-lddhpgl` with the `LDOPTS` environmental variable set to `-a archive`.

The reason for using the `LDOPTS` environmental variable is the `-l` option will look for a shared library driver first and then look for the archive driver if shared was not found. By exporting the `LDOPTS` variable as specified above, the `-l` option will only look for archive drivers. For more information, refer to the *Programming on HP-UX* manual on linking shared or archive libraries.

Note that if you link in `libddhpgl.a`, you must also link in `libdvio.a` (see the section **Examples**).

Examples

Assuming you are using `ksh`(1), to compile and link a C program for use with this driver, use:

```
export LDOPTS="-a archive"
```

and then:

```
cc example.c -L/usr/lib/X11R5 -lddhpgl -lXwindow\
-lsb1 -lsb2 -lXhp11 -lX11 -ldvio -lm  -o example
```

or for FORTRAN, use:

```
F77 example.f -Wl,-L/usr/lib/X11R5 -lddhpgl -lXwindow\
-lsb1 -lsb2 -lXhp11 -lX11 -ldvio -o example
```

or for Pascal, use:

```
pc example.p -Wl,-L/usr/lib/X11R5 -lddhpgl -lXwindow\
-lsb1 -lsb2 -lXhp11 -lX11 -ldvio -o example
```

Device Initialization

Parameters for gopen

The `gopen` procedure has four parameters: Path, Kind, Driver, Mode.

Path	The name of the special device file created by the **`mknod`** command specified in the last section (for example, **`/dev/hpgl`**.)
Kind	Indicates the I/O characteristics of the device. This parameter may be one of the following:

- `OUTDEV`—output only
- `INDEV`—input only
- `OUTINDEV`—input or output

Driver — The character representation of the driver type. This must be either **`hpgl`** or **`hpgls`**, e.g., on HP-IB devices:

`"hpgl"`	*for C.*
`'hpgl'//char(0)`	*for FORTRAN77.*
`'hpgl'`	*for Pascal.*

The following is an example on RS-232 devices:

`"hpgls"`	*for C.*
`'hpgls'//char(0)`	*for FORTRAN77.*
`'hpgls'`	*for Pascal.*

Mode — The mode control word, consisting of several flag bits *or* ed together. Listed below are the flag bits which have device-dependent actions:

`0`	open the device, but do nothing else.

`INIT`	open and initialize the device in a device-dependent manner. For plotters, `INIT` is a `DF` command. The following are not changed: ■ P1 and P2 ■ Current pen number and position ■ Pen speed, force and acceleration ■ 90 degree rotation or axis alignment
`RESET_DEVICE`	open and completely initialize the device. For plotters, this is an `IN` command. The values of P1 and P2 are set equal to the paper limits of the plotter.
`SPOOLED`	open the device for spooled operation. Only an `OUTDEV` may be spooled.
`THREE_D`	open the device and set Starbase to three-dimensional mode

Note Spooling with the HP-GL driver automatically scales P1 and P2 to the plotting surface area. In order to turn off the scaling function, the Starbase command `set_p1_p2` with `METRIC` units must be called.

Syntax Examples

For C Programs:

To open and initialize an HP-IB HP-GL device for output:

```
fildes = gopen("/dev/hpgl", OUTDEV, "hpgl", INIT);
```

To open and initialize an RS-232 HP-GL device for output:

```
fildes = gopen("/dev/plotter", OUTDEV, "hpgls", INIT);
```

For FORTRAN77 Programs:

To open an HP-IB HP-GL device for spooled output:

```
fildes = gopen('myfile'//char(0), OUTDEV, 'hpgl'//char(0), SPOOLED);
```

To open an RS-232 HP-GL device for spooled output:

```
fildes = gopen('myfile'//char(0), OUTDEV, 'hpgls'//char(0), SPOOLED);
```

For Pascal Programs:

To open and initialize an HP-IB HP-GL device for spooled output:

```
fildes := gopen('myfile', OUTDEV, 'hpgl', INIT+SPOOLED);
```

To open and initialize an RS-232 HP-GL device for spooled output:

```
fildes := gopen('myfile', OUTDEV, 'hpgls', INIT+SPOOLED);
```

Device Defaults

Color Table

The HP-GL default color table is the Starbase default color table. To read the current color table values, use the `inquire_color_table` procedure. The official color table is stored in the device driver, allowing different color tables to be used for different devices in the same program. The default color map has eight entries as shown in the table below:

Table 21-1. Default Color Table

Pen	Color	Red	Green	Blue
0	white (pen up)	0.0	0.0	0.0
1	black	1.0	1.0	1.0
2	red	1.0	0.0	0.0
3	yellow	1.0	1.0	0.0
4	green	0.0	1.0	0.0
5	cyan	0.0	1.0	1.0
6	blue	0.0	0.0	1.0
7	magenta	1.0	0.0	1.0

You can change the color tables values with the `define_color_table` procedure.

Red, Green and Blue Values

Functions that pass red, green and blue values are supported. The pen most closely corresponding in value to the red, green and blue values is selected using the current color table entries. A square-root-of-sum-of-squares algorithm is used to identify the pen.

Device Coordinate Origin Default

The device coordinate origin (0, 0) is device dependent. Use the hardware manual provided with your HP-GL device to get the range of device coordinate values and coordinate orientation.

Direct Output

The result of a `inquire_id` procedure call is the value returned by an `OI` command, i.e., the device is interrogated.

Echo Types

Both tracking and echo update use the current echo type as defined as follows:

Table 21-2. Current Echo Type

Type	Description
0	Pen Up
1	Pen Up
2	Pen Down

Line Type Defaults

The following table shows the predefined line types. Device dependent information is listed after the table.

Table 21-3. Predefined Line Types

Index	Name	Approximate Pattern
0	`SOLID`	Solid
1	`DASH`	0.25, 0.50, 0.25
2	`DOT`	4–8 dots per repeat length
3	`DASH_DOT`	0.4, 0.1, dot, 0.1, 0.35
4	`DASH_DOT_DOT`	0.35, 0.1, dot, 0.1, dot, 0.1, 0.35
5	`LONG_DASH`	0.375, 0.25, 0.375
6	`CENTER_DASH`	0.35, 0.1, 0.1, 0.1, 0.35
7	`CENTER_DASH_DASH`	0.25, 0.1, 0.1, 0.1, 0.1, 0.1, 0.25

HP-GL plotters do not support line type 4; line type 7 is substituted.

Number of Pens

The default number of pens is 8. The number of pens may be specified using the `HPGL_SET_PEN_NUM gescape`. The `gescape` commands unique to this device driver are discussed later in this section.

Plotter Units

If the device responds to an `OF` command, plotter units are set to that response. Otherwise, the plotter units parameter is set to a default value of 0.025 millimeter per plotter unit.

P1 and P2 Defaults

The values for P1 and P2 are device dependent. When you power up the plotter the values of P1 and P2 will equal the paper limits. Afterwards P1 and P2 will not change unless the user changes them from the plotter's front panel, the device is opened in `RESET_DEVICE` mode, or the Starbase command `set_p1_p2` is performed. If the paper size on the plotter is changed, it is the user's responsibility to ensure that the values of P1 and P2 are correct.

Spooled Output

The result of an `inquire_id` procedure call, when using spooled output, is always "HP-GL" (with a terminating `'\0'`).

The values used for P1, P2 and plotter resolution are the default values for the HP 7580B plotter with "D" size paper. A scaling command (HP-GL command `SC`) is automatically done to the spool file so that the default P1 and P2 are mapped onto the actual device. This means that the full VDC extent will be fitted to the plotting surface, and the entire picture will be plotted.

If the values of P1 and P2 are changed using the Starbase command `set_p1_p2` with `METRIC` units while in `SPOOLED` mode, the scaling will be turned *off*. The setting of P1 and P2 with `FRACTIONAL` units will *not* change the scaling.

Note Spooling with the HP-GL driver automatically scales P1 and P2 to the plotting surface area. To turn off the scaling function, the Starbase command `set_p1_p2` with `METRIC` units must be called.

If the Starbase command `set_p1_p2` with `METRIC` units is to be used while spooling, it *must* occur before any primitives are drawn or undesired results will occur.

Timeouts

A timeout of 10 seconds is used for the initial status read of the device (if not spooled), after which the timeout is 0 seconds (no timeout).

Starbase Functionality

Plotter Input

Each HP-GL plotter can be considered a locator device in digitizer mode. Three values are located: X, Y, and Z. The X and Y values specify an absolute Cartesian location on the plotter's scaled plotting area in Virtual Device Coordinates. The Z value equals the maximum Virtual Device Coordinate if the pen is down, and the minimum Virtual Device Coordinate if the pen is up.

When in digitizer mode, the plotter displays its "enter" indicator. The Enter button is used to trigger either an event or request.

Sample calls will not cause the plotter to display its enter indicator.

Note Not all plotters are capable of indicating an enter condition. Consult your plotter manual for further information.

HP 9111A/T Input

The HP 9111A/T Graphics Tablet can be considered a locator device and a choice device.

The 16 "soft keys" defined on the tablet can be used as choice input buttons.

The tablet's digitizing surface is the locator area. Three values are located: X, Y, and Z. The X and Y values specify an absolute Cartesian location on the tablet's surface. The location is in Virtual Device Coordinates. The Z value equals the maximum VDC if the stylus is pressed, and the minimum VDC if the stylus is not pressed.

Pen Selection

The following set of rules are used to select the pen the plotter will actually use.

If the program specifies a pen number that is zero, the plotter does a `PEN UP`.

If the program specifies a pen number that is less than or equal to the number of pens the device driver recognizes, that pen number is sent to the plotter. If the plotter has a pen with that number, it is used. If the plotter does not have a physical pen with that number, a device-dependent action will occur. Either the plotter will use the pen with the largest number, or a MOD calculation is made and the resulting pen number is used.

If the program specifies a pen number that is larger than the number of pens the device driver recognizes, the device driver does a MOD calculation to define the pen number to send to the plotter. If the MOD calculation returns a non-zero value, the driver sends that calculated pen number to the plotter. If the MOD calculation returns a zero value the device driver makes an exception from sending pen number 0, and sends the largest pen number.

Exceptions to Standard Starbase Support

Commands Not Supported (no-ops)

The following commands are not supported. If one of these commands is used by mistake, it will not cause an error.

```
alpha_transparency
await_retrace
backface_control
background_color
background_color_index
bank_switch
bf_alpha_transparency
bf_control
bf_fill_color
bf_interior_style
bf_perimeter_color
bf_perimeter_repeat_length
bf_perimeter_type
bf_surface_coefficients
bf_surface_model
bf_texture_index
bitmap_print
bitmap_to_file
block_move
block_read
block_write
clear_control
contour_enable
dbuffer_switch
dcbitmap_print
dcbitmap_to_file
dcblock_move
dcblock_read
dcblock_write
define_contour_table
define_raster_echo
define_texture
define_trimming_curve
deformation_mode
depth_cue
depth_cue_color
depth_cue_range
depth_cue-range
display_enable
double_buffer
drawing_mode
file_to_bitmap
file_to_dcbitmap
file_to_intbitmap
fill_dither
hidden_surface
intbitmap_print
intbitmap_to_file
intblock_move
intblock_read
intblock_write
interior_style (INT_OUTLINE)
interior_style (INT_POINT)
intline_width
light_ambient
light_attenuation
light_model
light_source
light_switch
line_endpoint
line_filter
pattern_define
perimeter_filter
set_capping_planes
set_model_clip_indicator
set_model_clip_volume
shade_mode
shade_range
surface_coefficients
surface_model
texture_index
texture_viewport
texture_window
viewpoint
write_enable
zbuffer_switch
```

Commands Conditionally Supported

The following commands are supported under the listed conditions:

`clear_view_surface`	New page on devices with automatic paper feeders.[1]
`define_color_table`	Updates software color table only (an operator must physically change the pens).
`hatch_spacing`	Care should be taken to specify spacings greater than or equal to one pen width.
`interior_style`	Only the `INT_SOLID`, `INT_HATCH`, and `INT_HOLLOW` styles are supported.
`text_precision`	Only `STROKE_TEXT` precision is supported.
`vertex_format`	The "`use`" parameter must be zero, any extra coordinates supplied will be ignored.
`with_data`	`partial_polygon_with_data3d` `polygon_with_data3d` `polyhedron_with_data` `polyline_with_data3d` `polymarker_with_data3d` `quadrilateral_mesh_with_data` `triangle_strip_with-data` Additional data will be ignored if not supported by this device. For example, contouring data will be ignored if this device does not support it.

[1] Some plotters will only eject the paper if it has been plotted on.

Parameters for gescape

The `hpgl` driver supports the following `gescapes`. Refer to Appendix A of this manual for details on gescapes.

- `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.

22

The CADplt Device Driver

Device Description

The CADplt Device Driver is an HP-GL command set driver. This driver is contained in the `libddCADplt.a` archive library or the `libddCADplt.sl` shared library. This driver provides hardware support for certain areas of functionality for Starbase graphics. All standard HP-GL command set devices should work properly with this driver. If CADplt does not work with your HP-GL plotter, try the HP-GL driver.

Hewlett-Packard has tested and supports the following HP-GL devices with HP-IB and serial RS-232 interfaces.

- HP 7510A color film recorder
- HP 7550A plotter
- HP 7570A plotter
- HP 7580B plotter†
- HP 7585B plotter†
- HP 7586B plotter
- HP 7595A plotter
- HP 7596A plotter
- HP C1600A plotter
- HP C1601A plotter
- HP 7575A plotter
- HP 7576A plotter

† For plotters with serial number 2402 or higher

Although this device driver and the `libddhpgl.a` device driver both access HP-GL devices, there are major differences between them. These differences are listed in the following table.

Table 22-1. CADplt and HPGL Driver Features

Feature	**CADplt**	**HP-GL**
Supports all HP-GL devices	no	yes
Supports input operations	no	yes
Hardware polygon support	yes	no
Hardware rectangle support	yes	no
Hardware text support (`FLOAT_XFORM` interface only)	yes	no
Roll paper support	yes	no
Isotropic spooling	yes	no
HP-GL error checking	yes	no

Setting Up the Device

Switch Settings

HP-IB Interfacing

The HP-IB address of the device must correspond to the device file minor number, see "Special Device Files (`mknod`)" in this chapter.

Serial RS-232 Interfacing

The serial interface on the device must be set as follows:

- 8-bit character size
- no parity
- desired baud rate
- one stop bit if baud rate is greater then 110, otherwise two stop bits

The device driver `libddCADplt.a` will automatically set the Operating System I/O interface for the serial device to the following configurations:

1. device handshaking
   - `XON/XOFF` protocol with `dc1` and `dc3` signals
   - "`;`" command terminator
   - ⟨*newline*⟩ response terminator
2. device interface, `termio`(4)
   - 8-bit character size
   - `XON/XOFF` protocol
   - no parity
   - disabled `INTR` and `QUIT` signals
   - 2400 baud rate if initially 300 [1]
   - no postprocessing
   - canonical processing
   - undefine `ERASE` and `KILL` symbols

[1] The default baud rate for a serial interface is 300 baud when the device file is freshly opened. If the default is still in effect, then the device driver will change the baud rate to 2400 as this is what many serial devices are run at. However, if you have changed the the baud rate from the default value of 300, then the driver assumes you have purposely changed it and will not modify it.

Note

There must *not* be a `getty` running on the serial device file. The following command will sleep a `getty`:

```
sleep 1000000 < /dev/plts &
```

Note

If the device is in `SPOOLED` mode, the device interface `termio`(4) will *not* be automatically configured for you. It is your responsibility to configure the interface correctly as below:

Given a freshly opened device interface with the following defaults:

```
300 cs8 cread hupcl
```

The following commands will correctly configure the device interface:

```
sleep 1000000 < /dev/plts &
stty ⟨baud⟩ ixon ignbrk icanon isig clocal < /dev/plts
stty erase ^- kill ^- < /dev/plts
```

where ⟨*baud*⟩ is the baud rate of the device and `/dev/plts` is the device file for the serial device.

Special Device Files (mknod)

The `mknod` command creates a special device file which is used to communicate between the computer and the peripheral device. See the `mknod`(1M) command in the *HP-UX Reference* manual for further information. The name of this special device file is passed to Starbase in the `gopen` procedure. Since superuser or root capabilities are needed to create special device files, they are normally created by the system administrator.

Although special device files can be made in any directory of the HP-UX file system, the convention is to create them in the `/dev` directory. Any name may be used for the special device file. The following examples will create a special device file for this device. Remember that you must be the superuser or root to use the `mknod` command.

For the Series 300 and 400

HP-IB Interface

The `mknod` parameters should create a character device file with a major number of 21 and a minor number of 0x⟨*sc*⟩⟨*ad*⟩00⟨ where ⟨*sc*⟩ is the select code and ⟨*ad*⟩ is the device's HP-IB address.

```
mknod /dev/plt c 21 0x⟨sc⟩⟨ad⟩00
```

Serial RS-232 Interface

The `mknod` parameters should create a character device file with a major number of 1 and a minor number of 0x⟨*sc*⟩⟨*ad*⟩04 where ⟨*sc*⟩ is the select code and ⟨*ad*⟩ is the port address.

```
mknod /dev/plts c 1 0x⟨sc⟩⟨ad⟩04
```

For the Series 700

Serial RS-232 Interface

For Serial Port A, the `mknod` parameters should create a character device file with a major number of 1 and a minor number of `0x204004`:

```
mknod /dev/plts c 1 0x204004
```

For Serial Port B, the `mknod` parameters should create a character device file with a major number of 1 and a minor number of `0x205004`:

```
mknod /dev/plts c 1 0x205004
```

Centronics Parallel Interface

The `mknod` parameters should create a character device file with a major number of 11 and a minor number of `0x206002`:

```
mknod /dev/plt_parallel c 11 0x206002
```

For the Series 800

HP-IB Card Device File

The `mknod` parameters should create a character device file with a major number of 21 and a minor number of `0x00`⟨*lu*⟩⟨*ad*⟩ where ⟨*lu*⟩ is the hardware logical unit and ⟨*ad*⟩ is the device's address.

```
mknod /dev/hpgl c 21 0x00⟨lu⟩⟨ad⟩
```

Serial Interface Card Device File

The `mknod` parameters should create a character device file with a major number of 1 and a minor number of `0x00` ⟨*lu*⟩ ⟨*ad*⟩ where ⟨*lu*⟩ is the hardware logical unit and ⟨*ad*⟩ is the port address.

```
mknod /dev/hpgl c 1 0x00⟨lu⟩⟨ad⟩
```

Linking the Driver

Shared Libraries

The shared device driver is the file named `libddCADplt.sl` in the `/usr/lib` directory. The device driver will be explicitly loaded at run time by compiling and linking with the starbase shared library `/usr/lib/libsb.sl`.

Note that use of the library `libdvio.a` requires the use of `-Wl,-E` when using plotter driver shared libraries as shown in the **Examples** section.

Examples

To compile and link a C program for use with the shared library driver, use:

```
cc example.c -L/usr/lib/X11R5 -Wl,-E -lddCADplt\
-lXwindow -lsb -lXhp11 -lX11 -ldvio -ldld -lm -o example
```

or with FORTRAN use,

```
F77 example.f -Wl,-L/usr/lib/X11R5 -Wl,-E -lddCADplt \
-lXwindow -lsb -lXhp11 -lX11 -ldvio -ldld -o example
```

or with Pascal use,

```
pc example.p  -Wl,-L/usr/lib/X11R5 -Wl,-E -lddCADplt\
-lXwindow -lsb -lXhp11 -lX11 -ldvio -ldld -o example
```

Upon device initialization the proper driver will be loaded. See the discussion of the `gopen` procedure in the **Device Initialization** section of this chapter for details.

Archive Libraries

The archive device driver is located in the `/usr/lib` directory with the file name `libddCADplt.a`. This device driver may be linked to a program by using the absolute path name `/usr/lib/libddCADplt.a`, an appropriate relative path name, or by using the `-l` option as in `-lddCADplt` with the `LDOPTS` environmental variable set to `-a archive`.

The reason for using the `LDOPTS` environmental variable is that the `-l` option will look for a shared library driver first and then look for the archive driver if shared was not found. By exporting the `LDOPTS` variable as specified above, the `-l` option will only look for archive drivers. For more information, refer to the *Programming on HP-UX* manual on linking shared or archive libraries.

Examples

Assuming you are using `ksh`(1), to compile and link a C program for use with this driver, use:

```
export LDOPTS="-a archive"
```

If you link in `libddCADplt.a`, you must also link in `libdvio.a` as below:

```
cc example.c -L/usr/lib/X11R5 -lddCADplt -lXwindow\
-lsb1 -lsb2 -lXhp11 -lX11 -ldvio -lm -o example
```

or for FORTRAN, use:

```
F77 example.f -Wl,-L/usr/lib/X11R5 -lddCADplt -lXwindow\
-lsb1 -lsb2 -lXhp11 -lX11 -ldvio  -o example
```

or for Pascal, use:

```
pc example.p -Wl,-L/usr/lib/X11R5 -lddCADplt -lXwindow\
-lsb1 -lsb2 -lXhp11 -lX11 -ldvio  -o example
```

Device Initialization

Parameters for gopen

The `gopen` procedure has four parameters: Path, Kind, Driver, Mode.

Path	This is the name of the special device file created by the **`mknod`** command as specified in the section “Special Device Files” such as `/dev/plt`.
Kind	This indicates the I/O characteristics of the device. This parameter may only be `OUTDEV`.
Driver	This is the character representation of the driver type. This must be `CADplt`.
Mode	This is the mode control word which consists of several flag bits which are *or* ed together. Listed below are the flag bits and their device dependent actions:

`0`	open the device but do nothing else
`INIT`	open and initialize the device in a device dependent manner. For this device driver the `INIT` mode will send the HP-GL command `DF` to the device. This command will *not* change the following: ■ P1 and P2 ■ pen speed, force and acceleration ■ 90 degree rotation or axis alignment
`RESET_DEVICE`	open and completely initialize the device. For this device driver the `RESET_DEVICE` mode will send the HP-GL command `IN` to the device. This command will reset the device’s configuration including P1 and P2.
`SPOOLED`	open the device for spooled operation.
`THREE_D`	open the device for three-dimensional primitives.

Syntax Example

For C programs:

```
fildes = gopen("/dev/plt", OUTDEV, "CADplt", RESET_DEVICE);

fildes = gopen("spoolfile", OUTDEV, "CADplt", RESET_DEVICE | SPOOLED);
```

For FORTRAN 77 programs:

```
  fildes = gopen('/dev/plt'//char(0), OUTDEV, 'CADplt'//char(0), INIT)

  fildes = gopen('/dev/plt'//char(0), OUTDEV, 'CADplt'//char(0), 0)
```

For Pascal programs:

```
 fildes := gopen('/dev/plt', OUTDEV, 'CADplt', RESET_DEVICE+THREE_D);
 fildes := gopen('spoolfile', OUTDEV, 'CADplt', RESET_DEVICE+SPOOLED);
```

Device Defaults

Color Table

The HP-GL default color table is the same as the Starbase default color table. To read the current color table values, use the `inquire_color_table` procedure. The official color table is stored in the device driver allowing different color tables to be used for different devices in the same program. The default color map has eight entries as shown in the following table.

Table 22-2. Default Color Map

Pen	Color	Red	Green	Blue
0	white (pen up)	0.0	0.0	0.0
1	black	1.0	1.0	1.0
2	red	1.0	0.0	0.0
3	yellow	1.0	1.0	0.0
4	green	0.0	1.0	0.0
5	cyan	0.0	1.0	1.0
6	blue	0.0	0.0	1.0
7	magenta	1.0	0.0	1.0

You can change the color table values with the `define_color_table` procedure.

Red, Green and Blue Values

Functions that pass red, green and blue values are supported. The pen most closely corresponding in value to the red, green and blue values is selected using the current color table entries. A "square root of sum of squares" algorithm is used to identify the pen.

Device Coordinates

The default number of millimeters per device coordinates is 0.025. If the `gopen` mode is *not* `SPOOLED`, the device driver will inquire the device and use the value returned.

Device Coordinate Origin

The device coordinate origin $(0,0)$ is device dependent. You should consult your device hardware manual to get the origin. In general, the origin is normally centered between the P1 and P2 extent so that there are an equal number of negative and positive device coordinates on each side of the origin. The one known exception is the HP 7550 plotter which places the origin in the lower left corner of the paper.

Device ID

If the device is *not* in SPOOLED mode, then the device driver will send the HP-GL command OI and use the returned string as the device ID. If the device is spooled, the device ID will be CADplt.

Line Types

The following table shows the default line types that are available.

Table 22-3. Line Types

Index	Type
0	SOLID
1	DASH
2	DOT
3	DASH_DOT
4	CENTER_DASH_DASH
5	LONG_DASH
6	CENTER_DASH
7	CENTER_DASH_DASH

Number of Pens

The default number of pens is 8. The number of pens may be specified using the HPGL_SET_PEN_NUMBER gescape.

P1 and P2

The values for P1 and P2 are device dependent and will vary depending on the gopen mode that was used when accessing the device as below:

- INIT or 0 mode

  The values of P1 and P2 will be equal to the current values the device is set to. The device driver will inquire these values and use them unmodified. When a device is opened in this mode, it is the your responsibility to insure that appropriate P1 and P2 values are currently established.

- RESET_DEVICE mode

The HP-GL command `IN` will be sent to the plotter. This will cause the device to reset P1 and P2 to take advantage of the full size of the paper that is currently loaded. The device driver will then inquire these values and use them. This mode insures that the current values of P1 and P2 will match the paper size that is loaded.

- `SPOOLED` mode

  Since the device driver cannot inquire the P1 and P2 values from the device, the driver assumes the limits are as below:

  ```
  P1 x: -23144, P1 y: -17048
  P2 x:  23144, P2 y:  17048
  ```

  which are the limits for HP 7596A 36-inch roll paper. The device driver will then put the HP-GL command `SC` in the spool file. This will cause the device to scale the assumed P1 and P2 values to the actual P1 and P2 values in effect when the spooled file is dumped to the device. The affect of the scaling command is to cause the entire drawing to be expanded or compressed so that it will fill the P1 and P2 extent that the device currently has. In order to turn off the scaling function, the Starbase procedure `set_p1_p2` with `METRIC` units must be called.

Note Some devices will not guarantee isotropic scaling when you spool to them. Check your device hardware manual to see if the HP-GL command `SC` supports the fifth parameter for isotropic scaling. If it does not, then the P1, P2 aspect ratio must match the default P1, P2 ratio above, or the drawing will be distorted.

Timeouts

An initial timeout of 10 seconds is used when the procedure `gopen` is called. If the device is accessed correctly by the `gopen` call within the timeout, then the timeout is removed completely for all further action. Should the device be taken off line or fail after a successful `gopen` call, the device driver can indefinitely “hang” during operation.

Starbase Functionality

Hardware Character Sets

When performing hardware generated text, this device driver will recognize the following character sets for the call `designate_character_set`. The device driver will then instruct the device to load that specific character set. If the designated character set is not supported, an error may or may not be reported according to the state of the buffer mode flag (see "Error Reporting and Buffer Mode" in this section). You should check the device hardware manual to see if the device will support the designated character set. At this time the device driver does *not* support variable width characters.

Note Hardware character sets are not supported when the device is `gopen`ed with `INT_XFORM`.

Table 22-4. Hardware Character Sets

Font #	CHSET name	Description
0	`usascii`	ANSI ASCII
1	`9825`	9825 Character Set
2	`french`	French
2	`german`	German
3	`scandinavian`	Scandinavain
4	`spanish`	Spanish/Latin American
5	`special`	Special Symbols
6	`jisascii`	JIS ASCII
7	`hproman`	Roman Extensions
8	`katakana`	Katakana
9	`iso_irv`	ISO Inter. Ref. Vers.
30	`iso_swedish_1`	ISO Swedish
31	`iso_swedish_2`	ISO Swedish for Names
32	`iso_norway_v1`	ISO Norway, Version 1
33	`iso_german`	ISO German
34	`iso_french_v1`	ISO French, Version 1

Table 22-4. Hardware Character Sets (continued)

Font #	CHSET name	Description
35	`iso_united`	ISO United Kingdom
36	`iso_italian`	ISO Italian
37	`iso_spanish`	ISO Spanish
38	`iso_portugues`†	ISO Portuguese
39	`iso_norway_v2`	ISO Norway, Version 2
60	`iso_french_v2`	ISO French, Version 2
99	`iso_drafting`	Drafting Symbols
100	`kanji_v1`	Kanji, part 1
101	`kanji_v2`	Kanji, part 2

†This is not a typographical error. The program recognizes this spelling.

Error Reporting and Buffer Mode

This device driver has two states for reporting errors depending on the buffer mode, as set by the procedure `buffer_mode`.

- Buffering On

  When buffering is enabled, the device driver will buffer all commands in an internal buffer before sending them to the device. All HP-GL errors generated by the device will be masked out. Regular Starbase errors will still be reported as normal.

- Buffering Off

  When buffering is disabled, the device driver will send each command to the device as it receives it. After each command is sent, the device driver will then inquire the device's status and report any HP-GL errors that occured. This mode should only be used when debugging an application.

When the `gopen` mode is `SPOOLED`, the spooled file will mask out all HP-GL errors generated, regardless of the buffer mode.

Hardware Polygon Support

All Starbase polygon interiors and borders are drawn by using the device's hardware support for polygons. This will normally result in an increase in rendering speed and a decrease in the size of spooled files. Polygon hardware support conforms to Starbase specifications as defined in the *Starbase Graphics Techniques* manual. Hardware support is provided through the use of the HP-GL commands `PM`, `FP`, and `EP`. You can *not* turn off hardware support of polygons.[2] The number of vertices supported is device dependent. For some devices, the default number of vertices supported can be modified by adjusting the size of the memory partitions through software control. There are two methods of changing the memory partition: through use of the HP-GL command `GM` and use of the HP-GL escape function "`ESC.T`". Users should refer to the device's programming manual on using these commands. The following table summarizes the default number of vertices supported and if that default can be changed using the HP-GL command `GM` or "`ESC.T`".

Table 22-5. Polygon Vertex Support

Device	# Vertices	GM	ESC.T
HP 7596A	219	yes	yes
HP 7595A	219	yes	yes
HP 7586B	218	no	yes
HP 7585B	218	no	yes
HP 7580B	218	no	yes
HP 7570A	93	yes	yes
HP 7550A	127	yes	yes
HP 7510A	495	yes	yes

[2] The present exception to this is for polygons drawn with the `interior_style` parameter `INT_HATCH`. At this time, hatching is performed only through software.

Table 22-5. Polygon Vertex Support (continued)

Device	# Vertices	GM	ESC.T
HP C1600A	1500†	no	no
HP C1601A	1500†	no	no
HP 7575A	93	yes	yes
HP 7576A	93	yes	yes

† The plotter has 16 900 bytes of available memory allocated to the downloadable character buffer as needed, the rest goes to the polygon buffer. For example, dividing 16 900 by 8 equals 2112.5. If you allow some extra for fill types, you can estimate that a polygon with up to 1500 points easily fits in the polygon buffer.

Hardware Rectangle Support

All Starbase rectangle interiors are drawn by using the device's hardware support for polygons. This will normally result in an increase in rendering speed and a decrease in the size of spooled files. Rectangle hardware support conforms to Starbase specifications as defined in the *Starbase Graphics Techniques* manual. Hardware support is provided through the use of the HP-GL commands `PM`, `FP` and `EP`. You can *not* turn off hardware support of rectangles. [3]

[3] The present exception to this is for polygons drawn with the `interior_style` parameter `INT_HATCH`. At this time, hatching is performed only through software.

Hardware Text Support

Starbase text can be drawn by using the device's hardware support for text. This support is conditional on use of the Starbase procedure `text_precision` with a precision parameter of `STRING_TEXT`. This will normally result in an increase in rendering speed, a decrease in the size of spooled files, and an increase in text quality. Since this support is user selectable, not all of those devices supported through this device driver support all those features of Starbase text. Differences between device hardware generated text and Starbase software generated text are listed below:

Table 22-6. Hardware Text Support

Starbase Call	Parameter	Group 1	Group 2	Group 3
`text_precision`	`STRING_TEXT`	yes	yes	no
`text_path`	`PATH_LEFT`	no	no	no
`text_path`	`PATH_UP`	no	no	no
`text_path`	`PATH_DOWN`	yes	no	no
`text_font_index`	$\langle index \rangle = 2$	no	no	no
`text_alignment`	`TA_CONTINUOUS_HORIZONTAL`	no	no	no
`text_alignment`	`TA_CONTINUOUS_VERTICAL`	no	no	no
`text_alignment`	`TA_CAP`	no	no	no
`text_alignment`	`TA_BASE`	no	no	no
`text_line_path`	(all)	no	no	no

- Group 1 = HP 7575A, HP 7576A.HP 7595A, HP 7596A.
- Group 2 = HP 7586B, HP 7585B, HP 7580B, HP 7550A, HP 7510A, HP C1600A, and HP C1601A.
- Group 3 = HP 7570A.

Pen Selection

If a program specifies a pen number that is larger then the number of pens the device has, the device driver will perform a "mod" [4]calculation to define the actual pen to be used. If the mod calculation returns a value of zero, then the largest pen number will be used instead.

If pen number 0 is selected, then a device dependent action will occur and you should consult your device hardware manual. In general, pen 0 will cause most devices to not select any pen at all when performing any drawing operation.

Roll Paper, Autoloading and Rasterizing

The device driver will attempt to set the paper size and perform a page feed using the HP-GL commands `PS` and `PG` when the Starbase procedure `gclose` is called. This will cause those devices using roll paper or having autoloading capabilities to feed the current drawing out. For those devices that accept HP-GL commands and then rasterize the data for output, this will cause the rasterization to occur and the drawing to be ejected. Devices supporting this functionality are shown below:

- HP 7596A
- HP 7586B
- HP 7550A
- HP 7510A
- HP C1600A
- HP C1601A

[4] The mod function is a remainder function. For example, 8 mod 3 = 2.

New Device Support

This driver uses a subset of the HP-GL command language. When attempting to use this device driver with unsupported devices, that device should support those HP-GL commands as required below:

Table 22-7. HP-GL Command Support

CM†	DF	DI†	DS†
DV†	EP‡	ES†	FP‡
IM	IN	IP	IV†
LB†	LO†	LT‡	OE
OF	OI	OP	PA‡
D‡	PG§	PM‡	PS§
PT‡	PU‡	SC	SR†
SL†	SP‡	VS	

† This command is only required for hardware text. If Starbase software generated text is used, the device does not need to support this command.

‡ This command is used for generating polygons, rectangles and lines. The device must implement this command for correct primitives.

§ This command is used for support of roll paper, autoloading and rasterizing devices.

Exceptions to Standard Starbase Support

Commands Not Supported (no-ops)

The following commands are not supported. If one of these commands is used by mistake, it will not cause an error.

```
alpha_transparency
await_retrace
backface_control
background_color
background_color_index
bank_switch
bf_alpha_transparency
bf_control
bf_fill_color
bf_interior_style
bf_perimeter_color
bf_perimeter_repeat_length
bf_perimeter_type
bf_surface_coefficients
bf_surface_model
bf_texture_index
bitmap_print
bitmap_to_file
block_move
block_read
block_write
clear_control
contour_enable
dbuffer_switch
dcbitmap_print
dcbitmap_to_file
dcblock_move
dcblock_read
dcblock_write
define_contour_table
define_raster_echo
define_texture
define_trimming_curve
deformation_mode
depth_cue
depth_cue_color
depth_cue_range
depth_cue-range
display_enable
double_buffer
drawing_mode
file_to_bitmap
file_to_dcbitmap
file_to_intbitmap
fill_dither
hidden_surface
intbitmap_print
intbitmap_to_file
intblock_move
intblock_read
intblock_write
interior_style (INT_OUTLINE)
interior_style (INT_POINT)
intline_width
light_ambient
light_attenuation
light_model
light_source
light_switch
line_endpoint
line_filter
pattern_define
perimeter_filter
set_capping_planes
set_model_clip_indicator
set_model_clip_volume
shade_mode
shade_range
surface_coefficients
surface_model
texture_index
texture_viewport
texture_window
viewpoint
write_enable
zbuffer_switch
```

Commands Conditionally Supported

The following commands are supported under the listed conditions:

`clear_view_surface`	Indicates a new page on devices with automatic paper feeders.[5]
`define_color_table`	updates software color table only (an operator must physically change the pens).
`hatch_spacing`	care should be taken to specify spacings greater than or equal to one pen width.
`interior_style`	only the `INT_SOLID`, `INT_HATCH`, and `INT_HOLLOW` styles are supported.

`with_data`	`partial_polygon_with_data3d`
	`polygon_with_data3d`
	`polyhedron_with_data`
	`polyline_with_data3d`
	`polymarker_with_data3d`
	`quadrilateral_mesh_with_data`
	`triangle_strip_with-data`
	Additional data will be ignored if not supported by this device. For example, contouring data will be ignored if the device does not support it.

`vertex_format`	the ⟨*use*⟩ parameter must be zero, any extra coordinates supplied will be ignored.

[5] Some plotters will only eject the paper if it has been plotted on.

Parameters for gescape

The `CADplt` driver supports the following gescapes. Refer to Appendix A of this manual for details on gescapes.

`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.

23

The CADplt2 Device Driver

Device Description

The driver archive library `libddCADplt.a` or shared library `libddCADplt.sl` contains the CADplt2 Device Driver as well as the CADplt Device Driver. The command plotter language for the CADplt2 driver is HP-GL/2.

The CADplt2 driver provides hardware support for certain areas of functionality for Starbase graphics. All standard HP-GL/2 command set devices should work properly with this driver. Hewlett Packard has tested and supports the following HP-GL/2 devices with HP-IB and serial RS-232 interfaces.

- HP C1600A B/W Electrostatic, D-size (HP 7600 Model 240D)
- HP C1601A B/W Electrostatic, E-size (HP 7600 Model 240E)
- HP 7595B DraftMaster SX (single sheet)
- HP 7596B DraftMaster RX (roll feed)
- HP 7599A DraftMaster MX (multi-user, roll or sheet)
- HP C1602A PaintJet XL with HP-GL/2 plug in cartridge
- HP C1620A Color Electrostatic (HP 7600 Model 355)
- HP C1625A B/W Electrostatic, US D-size (HP 7600 Model 250)
- HP C1627A B/W Electrostatic, US E-size (HP 7600 Model 255)
- HP C1629A B/W Electrostatic, EUROPE A1-size (HP 7600 Model 250)
- HP C1631A B/W Electrostatic, EUROPE A0-size (HP 7600 Model 255)

The following table displays the features of the CADplt2 driver.

Table 23-1. CADplt2 Device Driver Features

Feature	CADplt2
Supports all HP-GL devices	no
Supports input operations	no
Hardware polygon support	yes
Hardware rectangle support	yes
Hardware text support (`FLOAT_XFORM` interface only)	yes
Roll paper support	yes
Isotropic spooling	yes
HP-GL/2 error checking	yes
Starbase wide lines	yes
Encoded spool files	yes
HP-GL/2, PCL context switching	yes
Extended font selections	†yes
Single quadrant coodinate system	yes
Color map support	‡yes

† Supported if device contains desired fonts.
‡ Supported on color electrostatic plotters.

Setting Up the Device

Switch Settings

HP-IB Interfacing

The HP-IB address of the device must correspond to the device file minor number, see "Special Device Files (`mknod`)" in this chapter.

Serial RS-232 Interfacing

The serial interface on the device must be set as follows:

- 8-bit character size
- no parity
- desired baud rate
- one stop bit if baud rate is greater than 110, otherwise two stop bits

The `CADplt2` driver will automatically set the Operating System I/O interface for the serial device to the following configurations:

1. device handshaking
   - `XON/XOFF` protocol with `dc1` and `dc3` signals
   - "`;`" command terminator
   - ⟨*carriage return*⟩ response terminator—Serial RS-232 interface.
   - ⟨*carriage return*⟩⟨*line-feed*⟩ response terminator—HP-IB interface.

2. device interface, `termio`(4)
   - 8-bit character size
   - `XON/XOFF` protocol
   - no parity
   - disabled `INTR` and `QUIT` signals
   - 2400 baud rate if initially 300 [1]
   - no postprocessing
   - canonical processing
   - undefine `ERASE` and `KILL` symbols

[1] The default baud rate for a serial interface is 300 baud when the device file is freshly opened. If the default is still in effect, then the device driver will change the baud rate to 2400 as this is what many serial devices are run at. However, if you change the baud rate from the default value of 300, then the driver assumes you have purposely changed it and will not modify it.

Note

There must *not* be a `getty` running on the serial device file. The following command will sleep a `getty`:

```
sleep 1000000 < /dev/plts &
```

Note

If the device is in `SPOOLED` mode, the device interface `termio`(4) will *not* be automatically configured for you. It is your responsibility to configure the interface correctly as below:

Given a freshly opened device interface with the following defaults:

```
300 cs8 cread hupcl
```

The following commands will correctly configure the device interface:

```
sleep 1000000 < /dev/plts &
stty ⟨baud⟩ ixon ignbrk icanon isig clocal < /dev/plts
stty erase ^- kill ^- < /dev/plts
```

where ⟨*baud*⟩ is the baud rate of the device and `/dev/plts` is the device file for the serial device.

Special Device Files (mknod)

The `mknod` command creates a special device file which is used to communicate between the computer and the peripheral device. See the `mknod`(1M) command in the *HP-UX Reference* manual for further information. The name of this special device file is passed to Starbase in the `gopen` procedure. Since superuser or root capabilities are needed to create special device files, they are normally created by the system administrator.

Although special device files can be made in any directory of the HP-UX file system, the convention is to create them in the `/dev` directory. Any name may be used for the special device file. The following examples will create a special device file for this device. Remember that you must be the superuser or root to use the `mknod` command.

Series 300 and 400

HP-IB Interface

The `mknod` parameters should create a character device file with a major number of 21 and a minor number of `0x`⟨*sc*⟩⟨*ad*⟩`00`⟨ where ⟨*sc*⟩ is the select code and ⟨*ad*⟩ is the device's HP-IB address.

```
mknod /dev/plt c 21 0x⟨sc⟩⟨ad⟩00
```

Serial RS-232 Interface

The `mknod` parameters should create a character device file with a major number of 1 and a minor number of `0x`⟨*sc*⟩⟨*ad*⟩`04` where ⟨*sc*⟩ is the select code and ⟨*ad*⟩ is the port address.

```
mknod /dev/plts c 1 0x⟨sc⟩⟨ad⟩04
```

For the Series 700

Serial RS-232 Interface

For Serial Port A, the `mknod` parameters should create a character device file with a major number of 1 and a minor number of `0x204004`:

```
mknod /dev/plts c 1 0x204004
```

For Serial Port B, the `mknod` parameters should create a character device file with a major number of 1 and a minor number of `0x205004`:

```
mknod /dev/plts c 1 0x205004
```

Centronics Parallel Interface

The `mknod` parameters should create a character device file with a major number of 11 and a minor number of `0x206002`:

```
mknod /dev/plt_parallel c 11 0x206002
```

Series 800

HP-IB Card Device File

The `mknod` parameters should create a character device file with a major number of 21 and a minor number of `0x00`⟨*lu*⟩⟨*ad*⟩ where ⟨*lu*⟩ is the hardware logical unit and ⟨*ad*⟩ is the device's address.

```
mknod /dev/plt c 21 0x00⟨lu⟩⟨ad⟩
```

Serial Interface Card Device File

The `mknod` parameters should create a character device file with a major number of 1 and a minor number of `0x00` ⟨*lu*⟩ ⟨*ad*⟩ where ⟨*lu*⟩ is the hardware logical unit and ⟨*ad*⟩ is the port address.

```
mknod /dev/plts c 1 0x00⟨lu⟩⟨ad⟩
```

Linking the Driver

Shared Libraries

The shared device driver is the file named `libddCADplt.sl` in the `/usr/lib` directory. The device driver will be explicitly loaded at run time by compiling and linking with the starbase shared library `/usr/lib/libsb.sl`.

Note that use of the library `libdvio.a` requires the use of `-Wl,-E` when using plotter driver shared libraries as shown in the **Examples** section.

Examples

To compile and link a C program for use with the shared library driver, use:

```
cc example.c -L/usr/lib/X11R5 -Wl,-E -lddCADplt\
-lXwindow -lsb -lXhp11 -lX11 -ldvio -ldld -lm -o example
```

or with FORTRAN use,

```
F77 example.f -Wl,-L/usr/lib/X11R5 -Wl,-E -lddCADplt\
-lXwindow -lsb -lXhp11 -lX11 -ldvio -ldld -o example
```

or with Pascal use,

```
pc example.p  -Wl,-L/usr/lib/X11R5 -Wl,-E -lddCADplt\
-lXwindow -lsb -lXhp11 -lX11 -ldvio -ldld -o example
```

Upon device initialization the proper driver will be loaded. See the discussion of the `gopen` procedure in the **Device Initialization** section of this chapter for details.

Archive Libraries

The archive device driver is located in the `/usr/lib` directory in the library `libddCADplt.a`. This device driver may be linked to a program by using the absolute path name `/usr/lib/libddCADplt.a`, an appropriate relative path name, or by using the `-l` option as in `-lddCADplt` with the `LDOPTS` environmental variable set to `-a archive`.

The reason for using the `LDOPTS` environmental variable is that the `-l` option will look for a shared library driver first and then look for the archive driver if shared was not found. By exporting the `LDOPTS` variable as specified above, the

`-l` option will only look for archive drivers. For more information, refer to the *Programming on HP-UX* manual on linking shared or archive libraries.

Examples

Assuming you are using **ksh**(1), to compile and link a C program for use with this driver, use:

```
export LDOPTS="-a archive"
```

If you link in `libddCADplt.a`, you must also link in `libdvio.a` as below:

```
cc example.c -L/usr/lib/X11R5 -lddCADplt -lXwindow\
-lsb1 -lsb2 -lXhp11 -lX11 -ldvio -lm -o example
```

or for FORTRAN, use:

```
F77 example.f -Wl,-L/usr/lib/X11R5 -lddCADplt -lXwindow\
-lsb1 -lsb2 -lXhp11 -lX11 -ldvio -o example
```

or for Pascal, use:

```
pc example.p -Wl,-L/usr/lib/X11R5 -lddCADplt -lXwindow\
-lsb1 -lsb2 -lXhp11 -lX11 -ldvio -o example
```

Device Initialization

Parameters for gopen

The `gopen` procedure has four parameters: Path, Kind, Driver, Mode.

Path	This is the name of the special device file created by the `mknod` command as specified in the section "Special Device Files" such as `/dev/plt`.
Kind	This indicates the I/O characteristics of the device. This parameter may only be `OUTDEV`.
Driver	This is the character representation of the driver type. This must be `CADplt2`.
Mode	This is the mode control word which consists of several flag bits which are *or*ed together. Listed below are the flag bits and their device dependent actions:

`0`	open the device but do nothing else
`INIT`	open and initialize the device in a device dependent manner. For this device driver the `INIT` mode will send the HP-GL/2 command `DF` to the device. This command will *not* change the following: ■ P1 and P2 ■ media type and quality level ■ 90 degree rotation or axis alignment
`RESET_DEVICE`	open and completely initialize the device. For this device driver the `RESET_DEVICE` mode will send the HP-GL/2 command `IN` to the device. This command will reset the device's configuration including P1 and P2.
`SPOOLED`	open the device for spooled operation.
`THREE_D`	open the device for three-dimensional primitives.

Syntax Example

C programs:

```
fildes = gopen("/dev/plt", OUTDEV, "CADplt2", RESET_DEVICE);

fildes = gopen("spoolfile", OUTDEV, "CADplt2", RESET_DEVICE | SPOOLED);
```

FORTRAN 77 programs:

```
fildes = gopen('/dev/plt'//char(0), OUTDEV, 'CADplt2'//char(0), INIT)

fildes = gopen('/dev/plt'//char(0), OUTDEV, 'CADplt2'//char(0), 0)
```

Pascal programs:

```
fildes := gopen('/dev/plt', OUTDEV, 'CADplt2', RESET_DEVICE+THREE_D);

fildes := gopen('spoolfile', OUTDEV, 'CADplt2', RESET_DEVICE+SPOOLED);
```

PCL Context Switching

The CADplt2 driver can be used with devices that support HP-GL/2 and PCL (Printer Control Language). The driver will context switch the device into HP-GL/2 mode by sending the escape sequence $^{E}{}_{C}$%-IB before sending any HP-GL/2 commands. On devices which do not support PCL (pen plotters) the context switch command will be ignored.

Encoded Polyline Command (PE)

The CADplt2 driver makes extensive use of the HP-GL/2 command `PE`. This command provides move, draw, pen-up, pen-down, and select pen functionality in an encoded format. Spoolfile size is reduced depending on the mix of output primitives, and disc space is saved.

Device Defaults

Color Table

The HP-GL/2 default color table is the same as the Starbase default color table. The exception to this is that entry `0` is white (no pen) and entry `1` is black[2]. To read the current color table values, use the `inquire_color_table` procedure.

Color output results may differ depending on the device used. The color electrostatic plotter will achieve the truest color reproductions. It can reproduce a wide spectrum of colors since it has an arbitrary number of definable pens.

Black and white electrostatic plotters can only reproduce color map entries `0` for white and `1` for black. Any other color selection will result in either white or black.

Pen plotters may produce different results based on the colors the device has available. Pen plotters have a set number of physical pens. The color map should be resized and redefined to reflect the physical number of pens and pen colors in the following steps.

1. `gopen` the device.
2. Set the color map size using the `gescape` `HPGL2_SET_CMAP_SIZE` (Nine pens, 0 equals no pen, 1-8 are real pens).
3. Set the color map entries with the Starbase routine `define_color_table` to the red, green and blue values of the physical pens of the device.

Note If colors are selected by red, green, and blue values, Starbase will try to match the actual color map values as closely as possible.

[2] Electrostatic plotters can plot white (no pen) over an area already plotted in another color

The default color map has 64 entries with 17 shown in the following table (entries 18-63 are various color shades defined by Starbase).

Table 23-2. Default Color Map

Pen	Color
0	white
1	black
2	red
3	yellow
4	green
5	cyan
6	blue
7	magenta
8	10% gray
9	20% gray
10	30% gray
11	40% gray
12	50% gray
13	60% gray
14	70% gray
15	80% gray
16	90% gray
17	white

You can redefine the default color map size and contexts using the `gescape` `HPGL2_SET_CMAP_SIZE` and the Starbase routine `define_color_table`.

Defining, redefining, and sizing the color map will not increase the size of the spooled files.

Red, Green, and Blue Values

Functions that take red, green, and blue values as arguments are supported. Starbase chooses the pen that most closely corresponds in value to the red, green, and blue values selected using the color map entries and sends the color map index to the driver. A "square root of sum of squares" algorithm is used to identify the pen.

Each Starbase routine that selects color has two variants: (a) one takes a color map, and (b) the other takes a red, green, and blue triplet. See *Starbase Reference* manual for more information on color selection routines.

Device Coordinate System

HP-GL/2 is a single quadrant coordinate system, as opposed to HP-GL which is a four quadrant system. The default P1, P2 limits for the CADplt2 driver operating in this coordinate system are P1=0, 0, P2=35376, 24000, equal to the D-sized paper in a 7600/240D electrostatic plotter. Since plotter-unit size is not device dependent, these coordinates are correct for any HP-GL/2 plotter with D-sized paper.

- Non-spooled

  When opening the device directly (non-spooled), the driver will inquire the device's P1, P2 limits and use them unmodified. You *may* use `set_p1_p2` to change the P1, P2 limits.

- Spooled Mode

  If the device is opened in a spooled mode, the driver will put the plotter into scaled mode, isotropically scaling the D-sized coordinates into the maximum plotting area available. Again, clipping will be avoided. However, if you use `set_p1_p2` with the metric option while in spooled mode, the scaling will be turned off and clipping may result.

Device ID

If the device is *not* in `SPOOLED` mode, the device driver will send the HP-GL/2 command `OI` and use the returned string as the device ID. If the device is spooled, the device ID will be `CADplt2`.

Line Types

All the Starbase line types are supported in the CADplt2 driver. (Index 4, `DASH_DOT_DOT` not supported in the CADplt or HP-GL driver, is supported in the CADplt2 driver.)

The following table shows the default line types CADplt2 supports.

Table 23-3. Line Types

Index	Type
0	SOLID
1	DASH
2	DOT
3	DASH_DOT
4	DASH_DOT_DOT
5	LONG_DASH
6	CENTER_DASH
7	CENTER_DASH_DASH

The `gescape` `HPGL2_ADAPTIVE_LINES`[3] selects either fixed (default) or adaptive line types. An adaptive line type "fits" the pattern between endpoints to insure an integer number of patterns; thus, endpoints always have a line drawn to them. Fixed line types resemble lines on a raster display, where the pattern is not fitted but wrapped around the object. In this configuration, endpoints could show up in a "move" rather than "draw" region of the pattern.

P1 and P2

The values for P1 and P2 are device dependent and will vary depending on the `gopen` mode that was used when accessing the device as below:

- `INIT` or `0` mode

  The values of P1 and P2 will be equal to the current values the device is set to. The device driver will inquire these values and use them unmodified. When a device is opened in this mode, it is your responsibility to insure that appropriate P1 and P2 values are currently established.

- `RESET_DEVICE` mode

[3] Warning: Adaptive line types may produce solid-looking lines when used with primitives such as circles, which are rendered by using many small line segments. The pattern will "adapt" to each small line segment.

The HP-GL/2 command IN will be sent to the plotter. This will cause the device to reset P1 and P2 to the hard clip limits to take advantage of the full size of the paper that is currently loaded. The device driver will then inquire these values and use them. This mode insures that the current values of P1 and P2 will match the paper size that is loaded.

- SPOOLED mode

  Since the device driver cannot inquire the P1 and P2 values from the device, the driver assumes the limits as:

  ```
      P1 x:  0, P1 y:  0
      P2 x:  35376, P2 y:  24000
  ```

  The limits are the same for HP 7600/240D electrostatic plotter. The device driver will then put the HP-GL/2 command SC in the spool file. This will cause the device to scale the assumed P1 and P2 values to the actual P1 and P2 values in effect when the spooled file is dumped to the device. The affect of the scaling command is to cause the entire drawing to be expanded or compressed isotropically so that it will fill the P1 and P2 extent that the device currently has. In order to turn off the scaling function, the Starbase procedure set_p1_p2 with METRIC units must be called.

Note HP-GL/2 devices will scale isotropically, yielding no distortion of the plot in spooled mode.

Timeouts

An initial timeout of 10 seconds is used when the procedure gopen is called. If the device is accessed correctly by the gopen call within the timeout, the timeout is removed completely for all further action. Should the device be taken off line or fail after a successful gopen call, the device driver can indefinitely “hang” during operation.

Starbase Functionality

Hardware Character Sets

The CADplt2 driver supports hardware generated text through the Starbase `designate_character_set` subroutine. Check the device hardware manual to see if the device will support the designated character set. The recognized character set names appear in the following lists.

Note Hardware character sets are not supported when the device is `gopen`ed with `INT_XFORM`.

Table 23-4. Hardware Character Sets for CADplt2

ansi_8	hpkana8	iso16_portuguese
apl_bit	hpkatakana	iso17_spanish
apl_typewriter	hpkorean8	iso21_german
arabic	hplarge	iso25_french
ascii_cyrillic	hplatinspanish	iso2_irv
cyrillic	hplegal	iso57_chinese
default	hpline	iso60_norwegian
denmark_pc8	hpmath7	iso61_norwegian
ecma_latin1	hpmath8	iso69_french
hparabic8	hppi	iso84_portuguese
hpblock	hproman8	iso85_spanish
hpeurospanish	hpromanext	iso4_united
hpgerman	hpspanish	line_draw8
hpgl_download	hpthai8	norway_pc8
hpgl_drafting	turkish8	ocr-a
hpgl_symbols	iso10_swedish	ocr-b
hpgreek8	iso11_swedish	ocr-m
hphebrew7	iso13_katakana	oem_1
hphebrew8	usi14_jisascii	us_pc8
hphpl	iso15_italian	

Table 23-5.
Character Set Names Common
to the CADplt and CADplt2 drivers

usascii	katakana	iso_united
french	iso_irv	iso_italian
german	iso_swedish_1	iso_spanish
spanish	iso_swedish_2	iso_portugues†
special	iso_norway_v1	iso_norway_v2
jisascii	iso_german	iso_french_v2
hproman	iso_french_v1	iso_drafting

†This is not a typographical error. The program recognizes this spelling.

Note The following character set names are not available in the `CADplt2` driver, but are available in the `CADplt` driver:

```
9825
scandinavian
kanji_v1
kanji_v2
```

Typefaces

By using the `gescape` `HPGL2_FONT_TYPEFACE` you may select from the following list of font typespaces supported by HP-GL/2. The number in the left column is the `gescape` argument required to select that particular typeface.

Table 23-6. HP-GL/2 Typefaces

Argument	Typeface	Argument	Typeface
00	line_draw	43	itc_papf_chancery
01	pica	44	clarendon
02	elite	45	itc_zapf_dingbats
03	courier	46	cooper
04	helv	47	itc_bookman
05	tmsrmn	48	stick
06	letter_gothic	49	hpgl_drafting
07	script	50	hpgl_arc
08	prestige	51	gil_sans
09	caslon	52	univers
10	orator	53	bodini
11	presentations	54	rockwell
12	helv_condensed	55	melior
13	serifa	56	itc_tiffany
14	futura	57	itc_clearface
15	palatino	58	amelia
16	itc_souvenir	59	park_avenue
17	optima	60	handel_gothic
18	itc_garamond	61	dom_casual
19	cooper_black	62	itc_benguiat
20	ribbon	63	itc_cheltenham
21	broadway	64	century_expanded
22	bauer_bodini_condensed	65	franklin_gothic
23	century_schoolbook	66	franklin_gothic_condensed
24	university_roman	67	franklin_gothic_extra_condensed
25	helv_outline	68	plantin
26	futura_condensed	69	trump_mediaeval
27	itc_korinna	70	(available)
28	naskh	71	itc_american_typewriter

Table 23-6. HP-GL/2 Typefaces (continued)

Argument	Typeface	Argument	Typeface
29	`cloister_black`	72	`antique_olive`
30	`itc_galliard`	73	`antique_olive_compact`
31	`itc_avant_garde`	74	`itc_bauhaus`
32	`brush`	75	`century_oldstyle`
33	`blippo`	76	`itc_eras`
34	`hobo`	77	`friz_quadrata`
35	`windsor`	78	`itc_lubalin`
36	`helv_compressed`	79	`eurostile`
37	`helv_extra_compressed`	80	`eurostile_expanded`
38	`peignot`	81	`itc_serif_gothic`
39	`baskerville`	82	`signet_roundhand`
40	`itc_garamond_condensed`	83	`souvenir_gothic`
41	`trade_gothic`	84	`stymie`
42	`goudy_old_style`	85	`univers_condensed`

Error Reporting and Buffer Mode

The CADplt2 driver implements buffer mode by sending an output error command to the device during the `make_picture_current` driver entrypoint. If buffer mode is enabled, upper-level Starbase will automatically call `make_picture_current` after every logical set of output primitives, thus, checking for errors periodically.

Hardware Polygon Support

All Starbase polygon interiors and borders are drawn by using the device's hardware support for polygons. This will normally result in an increase in rendering speed and a decrease in the size of spooled files. Polygon hardware support conforms to Starbase specifications as defined in the *Starbase Graphics Techniques* manual. Hardware support is provided through the use of the HP-GL/2 commands `PM`, `FP`, and `EP`.

Consult your device's hardware manual for the actual number of vertices supported. You cannot turn off hardware support of polygons.[4] However, since the HP-GL/2 language specifies that hardware supporting HP-GL/2 must be able to draw polygons with at least 512 vertices, all polygons with greater than this number of vertices will be split in software. Once split, polygons of 512 or fewer vertices will be sent to the hardware to draw. Be aware that this process can be slow in the worst cases.

Hardware Text Support

Starbase text can be drawn by using the device's hardware support for text. This support is conditional on use of the Starbase procedure `text_precision` with a precision parameter of `STRING_TEXT`. This will normally result in an increase in rendering speed, a decrease in the size of spooled files, and an increase in text quality. Features supported by device hardware generated text and Starbase software generated text appear in the following table.

Table 23-7. Hardware Text Support

Starbase Call	Parameter	CADplt2
`text_precision`	`STRING_TEXT`	yes
`text_path`	`PATH_LEFT`	†yes
`text_path`	`PATH_UP`	†yes
`text_path`	`PATH_DOWN`	†yes
`text_font_index`	$\langle index \rangle$ = 1,2,4,6,8	‡yes
`text_alignment`	`TA_CONTINUOUS_HORIZONTAL`	no
`text_alignment`	`TA_CONTINUOUS_VERTICAL`	no
`text_alignment`	`TA_CAP`	no
`text_alignment`	`TA_BASE`	no
`text_line_path`	(all)	†yes

† See the table: Supported Combinations of `test_path` and `test_line_path`.
‡ See the table: Starbase Support of Font Typeface.

[4] The present exception to this is for polygons drawn with the `interior_style` parameter `INT_HATCH`. At this time, hatching is performed only through software.

Support of Starbase Font Typefaces

Starbase supports font typeface selection but only in a limited way. Through the call `text_font_index`, the index passed to the function indicates the following combinations of typeface, font spacing, and stroke weight. The CADplt2 driver supports the Starbase font indices in the following table; however, for a more extensive font typeface selection, use the `gescape` provided to access all the HP-GL/2 typefaces.

Table 23-8. Hardware Support of Text Font Indices

Font	Description
1	Stick font, fixed
2	Stick font, proportional
†4	Sans serif, proportional, normal stroke
†6	Sans serif, proportional, bold stroke
†8	Serif, proportional, bold stroke

† The HP-GL/2 language does not have sans serif or serif typefaces. Requesting sans serif will select Helv and serif will select Tms Rmn.

HP-GL/2 supports proportional fonts if they are present in the device. If a proportional font is selected, `inquire_text_extent` will return the bounding rectangle of a fixed font.

Supported Combinations of text_path, text_line_path

Other features of Starbase include `text_path` and `text_line_path`. Text path specifies which way to move the current position after each character. Text line path specifies the movement of the current position after a line-feed is encounted. Each path type has four possible values: up, down, left, and right. The following table is a quick reference for supported combinations of both text and line path.

Table 23-9.
Supported Combinations of
text_path and text_line_path

	text_line_path			
text_path	`path_right`	`path_left`	`path_up`	`path_down`
`path_right`	no	no	yes	yes
`path_left`	no	no	yes	yes
`path_up`	yes	yes	no	no
`path_down`	yes	yes	no	no

Pen Selection

See the section called "Color Table" under "Device Defaults" in this chapter for a detailed discussion about pen selection.

Roll Paper, Autoloading and Rasterizing

The device driver will attempt to set the paper size and perform a page feed using the HP-GL/2 commands `PS` and `PG` when the Starbase procedure `gclose` is called. This will cause those devices using roll paper or having autoloading capabilities to feed the current drawing out. Some devices use the `PG` command as a signal to begin rasterization. Devices supporting this functionality are shown below:

- HP C1600A
- HP C1601A
- HP C1600A B/W Electrostatic, D-size (HP 7600 Model 240D)
- HP C1601A B/W Electrostatic, E-size (HP 7600 Model 240E)
- HP C1620A Color Electrostatic (HP 7600 Model 355)
- HP C1625A B/W Electrostatic, US D-size (HP 7600 Model 250)
- HP C1627A B/W Electrostatic, US E-size (HP 7600 Model 255)
- HP C1629A B/W Electrostatic, EUROPE A1-size (HP 7600 Model 250)
- HP C1631A B/W Electrostatic, EUROPE A0-size (HP 7600 Model 255)

New Device Support

All of the following HP-GL/2 commands are used by the CADplt2 driver. When attempting to use this device driver with unsupported devices, be sure the device implements the same commands.

Table 23-10. HP-GL/2 Command Support

AD	IN	OP	RP
CR	IP	PC	SA
DF	LB	PE	SC
DI	LO	PG	SD
DV	LT	PM	SR
EC	MT	PS	SL
EP	NP	PW	SS
ES	OE	QL	VS
FP	OI		

Note The `PE` command encapsulates the functionality of the following commands: `PA`, `PR`, `PU`, `PD`, and `SP`; thus, they are no longer needed.

Exceptions to Standard Starbase Support

Commands Not Supported (no-ops)

The following commands are not supported. If one of these commands is used by mistake, it will not cause an error.

```
alpha_transparency              display_enable
await_retrace                   double_buffer
backface_control                drawing_mode
background_color                file_to_bitmap
background_color_index          file_to_dcbitmap
bank_switch                     file_to_intbitmap
bf_alpha_transparency           fill_dither
bf_control                      hidden_surface
bf_fill_color                   intbitmap_print
bf_interior_style               intbitmap_to_file
bf_perimeter_color              intblock_move
bf_perimeter_repeat_length      intblock_read
bf_perimeter_type               intblock_write
bf_surface_coefficients         interior_style (INT_OUTLINE)
bf_surface_model                interior_style (INT_POINT)
bf_texture_index                intline_width
bitmap_print                    light_ambient
bitmap_to_file                  light_attenuation
block_move                      light_model
block_read                      light_source
block_write                     light_switch
clear_control                   line_endpoint
contour_enable                  line_filter
dbuffer_switch                  pattern_define
dcbitmap_print                  perimeter_filter
dcbitmap_to_file                set_capping_planes
dcblock_move                    set_model_clip_indicator
dcblock_read                    set_model_clip_volume
dcblock_write                   shade_mode
define_contour_table            shade_range
define_raster_echo              surface_coefficients
define_texture                  surface_model
define_trimming_curve           texture_index
deformation_mode                texture_viewport
depth_cue                       texture_window
depth_cue_color                 viewpoint
depth_cue_range                 write_enable
depth_cue-range                 zbuffer_switch
```

Commands Conditionally Supported

The following commands are supported under the listed conditions:

`clear_view_surface`	Indicates a new page on devices with automatic paper feeders.[5]
`define_color_table`	Updates software color table only. An operator must physically change the pens on a pen plotter—color electrostatic plotters support this fully.
`hatch_spacing`	Care should be taken to specify spacings greater than or equal to one pen width.
`interior_style`	Only the `INT_SOLID`, `INT_HATCH`, and `INT_HOLLOW` styles are supported.
`vertex_format`	The ⟨*use*⟩ parameter must be zero, any extra coordinates supplied will be ignored.
`with_data`	`partial_polygon_with_data3d` `polygon_with_data3d` `polyhedron_with_data` `polyline_with_data3d` `polymarker_with_data3d` `quadrilateral_mesh_with_data` `triangle_strip_with-data` Additional data per vertex will be ignored if not supported by this device. For example, contouring data will be ignored if the device does not support it.

[5] Some plotters will only eject the paper if it has been plotted on.

Parameters for gescape

The gescape functions are discussed in detail in the appendix of this manual.

The following gescape functions are common to two or more drivers:

HPGL_READ_BUFFER	Allows you to read data from the device.
HPGL_SET_PEN_SPEED	Allows you to change pen velocity.
HPGL_WRITE_BUFFER	Permits direct communication of HP-GL/2 commands to supported devices.

The following gescape functions are unique to this driver:

HPGL2_ADAPTIVE_LINES	Determines adaptive or fixed line types.
HPGL2_CUTTER_CONTROL	Enable/disable paper cutter.
HPGL2_FONT_POSTURE	Indicates upright or italic font posture.
HPGL2_FONT_TYPEFACE	Selects typeface.
HPGL2_FONT_WEIGHT	Sets the font stroke weight independent of Starbase.
HPGL2_LOGICAL_PEN_WIDTH	Determines the logical pen width.
HPGL2_REPLOT	Indicates number of replots for the command buffer.
HPGL2_SET_CMAP_SIZE	Indicates the size of the color map: number of pens available.
HPGL2_SET_MEDIA_TYPE	Determines the type of media to be used.
HPGL2_SET_QUALITY	Indicates the quality level of the output.

24

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)
   HP C1645A (PaintJet XL-300, PCL-5 mode)

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), HP-UX Release 1.2 and later on the Series 800, and HP-UX release 8.05 and latter on the Series 700. 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/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/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. Please refer to the information on how to create `dev` files for printers in the HP-UX manual *Installing Peripherals*.

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 nonzero 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 24-1. 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:

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 downloading 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.

‡ With B-sized paper.

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).

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 (Halftoning)

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`).
- select a single plane to print (`-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 `-ldd300l`. 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  \
-ldd300l -lsb1 -lsb2 -lm -o myprog

fc myprog.f /usr/lib/starbase/formatters/fmt_table.c \
/usr/lib/starbase/formatters/pcl/libfmtpcl.a \
-ldd300l -lsb1 -lsb2 -lm -o myprog

pc myprog.p /usr/lib/starbase/formatters/fmt_table.c \
/usr/lib/starbase/formatters/pcl/libfmtpcl.a \
-ldd300l -lsb1 -lsb2 -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. All Series 700 releases support the LP spooler.
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 Series 300/400 `mknod` for an HP 3630A at select code 9

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

2. To configure the port for normal printing

   ```
   stty  -parenb -ienqak 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
   ```

FINAL TRIM SIZE : 7.5 in x 9.0 in

25

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 extensions 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 a Starbase bitmap file to a color or monochromatic 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 only the display's image planes and current hardware color map. Note that in X windows applications, the overlay planes will not be printed.

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

Series 300:	`hp300l`, `hp300h`, `hp98550`, `hp98556`, `hp98704`, `hp98705`, `hp98720`, `hp98721`, `hp98730`, `hp98731`. `hp98735`, `hp98736`, `hpa1096`.
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 following devices work in conjunction with the PCL Imaging Formatter:

   HP C1602A (PaintJet XL).
   HP C1645A (PaintJet XL-300 in PCL5 mode).

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 algorithms:

   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, 8.05 and later for Series 700 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 printers 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 myprint.bit > /dev/rlp");
```

- Print currently displayed bitmap from a program in color:

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

- Print currently displayed window from a program in color:

```
/*Origin=10,10, Width=100, Height=200 */
system("screenpr -I -X10 -Y10 -D100 -H200\
-F/dev/crt > /dev/rlp");
```

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

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

- Print currently displayed bitmap in grayscale, rotated:

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

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 myprint.bit | lp -oraw");
```

- Print currently displayed bitmap from a program in color:

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

- Print currently displayed window from a program in color:

```
/* Origin=10,10, Width=100, Height=200 */
system("screenpr -I -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 grayscale, 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:

```
/usr/bin/pcltrans

/usr/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). The 8.07 release and latter Series 700 releases have the following device files already created:

```
/dev/ptr_parallel
/dev/plt_rs232_a
/dev/plt_rs232_b
```

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). Please refer to the information on how to create `dev` files for printers in the HP-UX manual *Installing Peripherals*.

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 PCL Imaging Formatter supports the following PCL printers:

Printer	Dots Per Inch	Paper Size	Starting Release
PaintJet XL (C1602A)	180	A or B	7.0
PaintJet XL-300 (C1645A, PCL-5 mode)	300	A	8.07 (Series 700) 8.0 (Series 300/400)

Print Modes

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

Table 25-1. Print Modes

Selection	Algorithm
0	no algorithm
1	snap to primaries
2	snap to black and 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).

Snap to Black and White

The `monochrome` mode maps each non-zero 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 non-white 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` to orient the print across the width of the paper by using the `-R` option.

Print Size 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`, and `-h` options of the `pcltrans` and `screenpr` HP-UX commands.

Paper size is automatically sensed by the PaintJet XL printer.

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

Warning and Error Messages

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

FINAL TRIM SIZE : 7.5 in x 9.0 in

26

Computer Graphics Metafile

Introduction

This section describes the CGM driver, which produces an ANSI/ISO standard Computer Graphics Metafile (CGM). The CGM is a metafile for capturing and storing device independent picture descriptions. It may contain multiple pictures. For further details and examples about the CGM driver, you may refer to the *Starbase Concepts and Tutorials* manual.

There also exists a CGM interpreter that reads a CGM and outputs to graphics devices. See `cgm_to_starbase`(3g) in the *Starbase Reference*. For an overview of CGM, see the CGM chapter of the *Starbase Graphics Techniques*.

Functionality and Encodings

The CGM standard defines nineteen primitives (lines, markers, text, circles, etc.) and thirty-five primitive attributes (text color, line pattern, interior style, etc.) for describing the contents of pictures. The CGM standard describes these capabilities in an abstract manner and defines three methods of encoding the elements. The `hpcgm` device driver supports the following three encoding methods (see also "Parameters for `gescape`," `CGMESC_ENCODING` later in this driver).

- The Binary encoding is reasonably compact and is optimized for CPU efficiency in generating and interpreting CGMs, but it is not human readable and may cause difficulties in some communications environments.
- The Clear Text encoding is human readable (for example, `CIRCLE (573,721) 95;`) and can be produced with a normal text editor. It is good for debugging and quick demonstrations but is not compact. It is relatively inefficient for CPUs to generate and interpret code using this method.

- The Character encoding method codes all data as ASCII characters. It is compact and good for communications, and probably lies between the Binary and Clear Text in CPU efficiency.

More information on CGM may be found in the ANSI X3.122-1986 and ISO 8632/1-4.

Precisions

The CGM defines elements for varying the precisions, types, and modes of data in a metafile. The `hpcgm` driver encodes coordinate data as type integer, and allows selection of low or high precision (16 bits or 32 bits per coordinate) See "Parameters for `gescape`," `CGMESC_VDC_PREC` later in this driver).

Mode

The CGM allows such things as marker size (as well as line width and edge width) to be expressed in one of two modes: scaled or absolute. Absolute mode means that size (width) is measured in coordinate units. Scaled mode means that the given size is a scale factor to be applied to the nominal marker size of the device upon which the CGM is displayed. CGM only allows one mode per picture. The `hpcgm` driver uses scaled mode. Any absolute sizes received from Starbase are converted to a scale factor.

The CGM standard also allows color to be selected either by index into a table (and provides a color table definition element) or by an RGB (Red, Green, Blue) triple. The `hpcgm` driver maps all Starbase color requests into RGB triples.

Picture

A CGM consists of one or more logically independent pictures. A picture consists of the graphical actions that occur between Starbase `clear_view_surface` calls. The `hpcgm` driver responds to a `clear_view_surface` call by terminating the current picture and initiating a new picture.

To Compile and Link with the Device Driver

For Shared Libraries

The compiler driver programs (`cc, fc, pc`) link with shared libraries by default. The shared device driver is the file named `libddhpcgm.sl` in the `/usr/lib` directory. Starbase will explicitly load the device driver at run time when you compile and link with the Starbase shared library `/usr/lib/libsb.sl`, or use the `-lsb` option. This loading occurs at `gopen`(3G) time.

Examples

To compile and link a C program for use with the shared library driver, use:

```
cc example.c -I/usr/include/X11R5/x11\
-L/usr/lib/X11R5 -lXwindow -lsb\
-lXhp11 -lX11 -ldld -lm -o example
```

or with FORTRAN use,

```
F77 example.f -Wl,-L/usr/lib/X11R5 -lXwindow -lsb\
-lXhp11 -lX11 -ldld -lm  -o example
```

or with Pascal use,

```
pc example.p  -Wl,-L/usr/lib/X11R5 -lXwindow -lsb\
-lXhp11 -lX11 -ldld -lm -o example
```

For details, see the discussion of the `gopen` procedure in the section *To Open and Initialize the Device* in this chapter.

For Archive Libraries

The archive device driver is located in the `/usr/lib` directory with the file name `libddhpcgm.a`.

You can link this device driver to a program by using any one of the following:

1. the absolute path name `/usr/lib/libddhpcgm.a`
2. an appropriate relative path name
3. the `-lddhpcgm` option with the `LDOPTS` environmental variable exported and set to `"-a archive"`.

By default, the linker program `ld`(1) looks for a shared library driver first and then the archive library driver if a shared library was not found. By exporting the `LDOPTS` variable, the `-l` option will refer only to archive drivers.

Examples

Assuming you are using `ksh`(1), to compile and link a C program for use with this driver, use:

```
export LDOPTS="-a archive"
```

and then:

```
cc example.c -lddhpcgm -L/usr/lib/X11R5 -lXwindow\
-lsb1 -lsb2 -lXhp11 -lX11 -lm  -o example
```

or for FORTRAN, use:

```
F77 example.f -lddhpcgm -Wl,-L/usr/lib/X11R5 -lXwindow\
 -lsb1 -lsb2 -lXhp11 -lX11 -o example
```

or for Pascal, use:

```
pc example.p -lddhpcgm -Wl,-L/usr/lib/X11R5 -lXwindow\
-lsb1 -lsb2 -lXhp11 -lX11 -o example
```

Initialization

Parameters for gopen

The `gopen` procedure has four parameters: Path, Kind, Driver, and Mode.

Path	The name of the file that will be created by Starbase and to which `hpcgm` will write the metafile.
Kind	May be `OUTMETA` or `OUTDEV`. If `OUTDEV`, the file named by path must already exist unless `SPOOLED` is specified in Mode.
Driver	The character representation of the driver type. This is `hpcgm` modified to meet the syntax of the programming language used. Namely:

`"hpcgm"`	*for C.*
`'hpcgm'//char(0)`	*for Fortran77.*
`'hpcgm'`	*for Pascal.*

Mode — The mode control word consisting of several flag bits that can be *or*ed together. Listed below are the flag bits which have device dependent action.

- `SPOOLED`—Allows specifying Kind equal to `OUTDEV` without having Path already in existence.
- `0` (zero)—No flag causes the device to be initialized anyway (including color map initialization).

Syntax Examples

The following examples open and initialize the `hpcgm` driver and put the metafile into a file named `example.cgm`:

For C Programs:

```
fildes = gopen("example.cgm", OUTMETA, "hpcgm", INIT);
```

For Fortran77 Programs:

```
fildes = gopen('example.cgm'//char(0), OUTMETA, 'hpcgm'//char(0), INIT);
```

For Pascal Programs:

```
fildes = gopen('example.cgm', OUTMETA, 'hpcgm', INIT);
```

Driver Default

There are a number of driver options that may be manipulated with the `gescape` function. See the “Parameters for `gescape`” section in this driver for the defaults and options.

Default Color Map

While the `hpcgm` driver produces a metafile with color selection mode `direct` (RGB). It also maintains an internal color map to convert indexes to RGB. This map has 256 entries and is initialized to the default values shown below.

Table 26-1. Default Color Table

Index	Color	Red	Green	Blue
0	black	0.0	0.0	0.0
1	white	1.0	1.0	1.0
2	red	1.0	0.0	0.0
3	yellow	1.0	1.0	0.0
4	green	0.0	1.0	0.0
5	cyan	0.0	1.0	1.0
6	blue	0.0	0.0	1.0
7	magenta	1.0	0.0	1.0
8	10% gray	0.1	0.1	0.1
9	20% gray	0.2	0.2	0.2
10	30% gray	0.3	0.3	0.3
11	40% gray	0.4	0.4	0.4
12	50% gray	0.5	0.5	0.5
13	60% gray	0.6	0.6	0.6
14	70% gray	0.7	0.7	0.7
15	80% gray	0.8	0.8	0.8
16	90% gray	0.9	0.9	0.9
17	white	1.0	1.0	1.0
18-255	shaded colors			

Selection of TOP mode (see `CGMESC_TOP_MODE gescape` later in this driver) changes the value of the default color table.

Table 26-2. Top Mode Default Color Table

Index	Color	Red	Green	Blue
0	black	0.0	0.0	0.0
1	white	1.0	1.0	1.0
2	red	1.0	0.0	0.0
3	green	0.0	1.0	0.0
4	blue	0.0	0.0	1.0
5	yellow	1.0	1.0	0.0
6	magenta	1.0	0.0	1.0
7	cyan	0.0	1.0	1.0
8–255	repeat colors†			

† Index numbers 8 through 255 repeat the colors listed in index 0–7.

When `INIT` is used in the `shade_mode` procedure call, the color map initialization is based on the value of the mode parameter.

`CMAP_NORMAL` mode	Same as the Default Color Table.
`CMAP_MONOTONIC` mode	The color map is initialized as shades of gray.
`CMAP_FULL` mode	The color map is initialized as shades of color with three bits allocated for red, three bits allocated for green, and two bits allocated for blue.

Starbase Functionality

Commands Not Supported (no-ops)

The following Starbase commands are not supported and are ignored.

```
alpha_transparency
await_retrace
backface_control
bank_switch
bf_alpha_transparency
bf_control
bf_fill_color
bf_interior_style
bf_perimeter_color
bf_perimeter_repeat_length
bf_perimeter_type
bf_surface_coefficients
bf_surface_model
bf_texture_index
block_move
block_read
block_write
clear_control
contour_enable
dbuffer_switch
dcblock_move
dcblock_read
dcblock_write
dcecho_type
dcecho_update
define_contour_table
define_raster_echo
define_texture
define_trimming_curve
deformation_mode
depth_cue
depth_cue_color
depth_cue_range
display_enable
double_buffer
drawing_mode
echo_type
echo_update
fill_dither
hidden_surface
inqiure_pick_depth
inquire_hit
inquire_pick_window
interior_style (INT_OUTLINE)
interior_style (INT_POINT)
light_ambient
light_attenuation
light_model
light_source
light_switch
line_filter
pattern_define
perimeter_filter
set_capping_planes
set_hit_mode
set_model_clip_indicator
set_model_clip_volume
set_pick_depth
set_pick_window
shade_range
surface_coefficients
surface_model
texture_index
texture_viewport
texture_window
track
track_off
viewpoint
write_enable
zbuffer_switch
```

Conditionally Supported

The following Starbase commands are supported under the listed conditions:

`clear_view_surface`	This causes completion of a previous picture and begins a new picture in metafile.
`shade_mode`	The color map mode is selected but shading cannot be turned on.
`vertex_format`	The use parameter must be zero. Any extra coordinates that are supplied are ignored.

`with_data`	`partial_polygon_with_data3d`
	`polygon_with_data3d`
	`polyhedron_with_data`
	`polyline_with_data3d`
	`polymarker_with_data3d`
	`quadrilateral_mesh_with_data`
	`triangle_strip_with-data`
	Additional data per vertex will be ignored if not supported by this device. For example, contouring data will be ignored if the device does not support it.

Parameters for gescape

The `hpcgm` driver recognizes a number of `gescape` functions. Following are the supported functions and definition of when they may be invoked.

After `gopen`, but before any other graphical activity:

- `CGMESC_ENCODING`—Selects CGM encoding.
- `CGMESC_MET_NAME`—Defines metafile name.
- `CGMESC_TOP_MODE`—Selects TOP mode for metafile generation.
- `CGMESC_VDC_PREC`—Selects VDC integer precision.

Anytime after `gopen`:

- `CGMESC_APPL_DATA`—Generates CGM application data element.
- `CGMESC_ESCAPE_ELT`—Generates CGM escape element.
- `CGMESC_FONT_IX`—Allows application to select fonts.
- `CGMESC_MESSAGE`—Generates CGM message element.
- `CGMESC_PIC_NAME`—Defines picture name.

The `gescape` function allows the application program to input or output to a device in a device dependent manner. The syntax for the `gescape` function is:

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

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

The ⟨*op*⟩ is the opcode that specifies the action to be performed.

The `arg1` and `arg2` parameters provide information needed by a `gescape`.

A detailed discussion on each of the `gescape` functions can be found in the appendix of this manual.

27 The HP Starbase-to-Visualizer Archive Device Driver

Device Description

This device driver accepts Starbase function calls and encodes acceptable 3-D display geometry into a special geometry metafile. This archive file serves as an intermediate representation of Starbase data destined to be rendered in the Personal Visualizer interactive environment. Additional processing by a support translator must be performed before the information is suitable for the Personal Visualizer.

Functionality

The HP Starbase-to-Visualizer `hpsbv` device driver is designed to facilitate data exportation from existing Starbase applications to the Personal Visualizer or Advanced Visualizer platform. Graphics code designed to display Starbase geometry primitives (such as polygons) can be adapted to output this information as an archive file which preserves the three dimensional spatial relationships of the user's database. The information stored in this file is then converted by a secondary translator (`sbvtrans`) to object data importable by the Personal Visualizer.

The `hpsbv` device driver functions as a virtual 3-D display device (with the Personal Visualizer as the actual display). It captures all view-independent matrix transformations to the archive file while ignoring calls for view points, clipping planes, and display-dependent information. Transformations normally associated with the interactive viewing process are then reserved for use by the Personal Visualizer.

The Starbase-to-Visualizer pipeline to the Personal Visualizer supports only geometry (polygons, spline surfaces, ellipses, etc.) and geometric information (such as vertex normals). Attribute information such as color, surface properties, and rendering-dependent data are not accessible by the Personal Visualizer. Using driver specific graphics switches (`gescape` functions), however, data organization in the archive can make color specification in the Personal Visualizer nearly comparable to rendering in a Starbase application. (See the `gescape` `SBVESC_OBJ_NAME` description later in this chapter for more information about this approach.)

Object Definitions

The Personal Visualizer deals with data as objects in a 3-D environment. Objects are composed of one or more polygonal faces. The result of translating an HPSBV archive file is one object file containing all geometry collected by the graphics driver. The name of this object is derived from the name of the archive unless another is specified via a graphics escape. (Special gescapes can be used to name multiple object definitions in the same output archive).

Linking the Driver

Shared Libraries

The shared HP Starbase-to-Visualizer Archive device driver is the file named `libddsbv.sl` in the `/usr/lib` directory. The device driver will be explicitly loaded at run time by compiling and linking with the Starbase Shared Library `/usr/lib/libsb.sl`, or by using the `-l` option `-lsb`.

Examples

To compile and link a C program for use with the shared library driver, use:

```
cc example.c -I/usr/include/X11R5/x11 -L/usr/lib/X11R5\
-lXwindow -lsb -lXhp11 -lX11 -ldld -lm -o example
```

or with FORTRAN use,

```
F77 example.f -Wl,-L/usr/lib/X11R5 -lXwindow -lsb\
-lXhp11 -lX11 -ldld -lm  -o example
```

or with Pascal use,

```
pc example.p  -Wl,-L/usr/lib/X11R5 -lXwindow -lsb\
-lXhp11 -lX11 -ldld -lm -o example
```

For details, see the discussion of the `gopen` procedure in the section *To Open and Initialize the Device* in this chapter.

Archive Libraries

The `hpsbv` device driver is the file named `libddsbv.a` in the `/usr/lib` directory. This device driver may be linked to a program using the absolute path name `/usr/lib/libddsbv.a`, an appropriate relative path name, or by using the `-l` option `-lddsbv` with the `LDOPTS` environmental variable set to `-a archive`.

The reason for using the `LDOPTS` environmental variable is that the `-l` option will look for a shared library driver first and then look for the archive driver if shared was not found. By exporting the `LDOPTS` variable as specified above, the `-l` option will only look for archive drivers. For more information, refer to the *Programming on HP-UX* manual on linking shared or archive libraries.

This driver also requires the math library to be linked with C programs. All programs must also be linked with the Starbase graphics libraries `/usr/lib/libsb1.a` and `/usr/lib/libsb2.a`, or use the `-l` option `-lsb1` and `-lsb2` . The device driver needs to precede the graphics libraries when linking, as shown in the examples below.

Examples

Assuming you are using `ksh`(1), to compile and link a C program for use with this driver, use:

```
export LDOPTS="-a archive"
```

and then:

```
cc example.c -lddsbv -L/usr/lib/X11R5 -lXwindow\
-lsb1 -lsb2 -lXhp11 -lX11 -lm  -o example
```

or for FORTRAN, use:

```
F77 example.f -lddsbv -Wl,-L/usr/lib/X11R5 -lXwindow\
 -lsb1 -lsb2 -lXhp11 -lX11 -o example
```

or for Pascal, use:

```
pc example.p -lddsbv -Wl,-L/usr/lib/X11R5 -lXwindow\
 -lsb1 -lsb2 -lXhp11 -lX11 -o example
```

Device Initialization

Parameters for `gopen`

The gopen procedure has four parameters: Path, Kind, Driver, and Mode.

- **Path** - The name of the archive file which will be created by Starbase and to which `hpsbv` will write the metafile. It is recommended that pathnames end with the extension `.sbv` for easy file administration.
- **Kind** - Must be `OUTDEV`, with `SPOOLED` specified in the **Mode** argument.
- **Driver** - -The character representation of the driver type. This is `hpsbv` modified to meet the syntax of the programming language used. Namely:

`"hpsbv"`	*for C*
`'hpsbv'//char(0)`	*for FORTRAN77*
`'hpsbv'`	*for Pascal*

- **Mode** - The *mode control word* consists of several flag bits that must be OR'ed together. Listed below are the flag bits which have device dependent action. These flags must be present to initialize the `hpsbv` driver.

`SPOOLED`	Allows specifying Kind equal to `OUTDEV` without having **Path** already in existence.
`INIT`	Open and initialize the device.

Syntax Examples

The following examples open and initialize the `hpsbv` driver and put the archive file into a file named `example.sbv`

For C Programs:

```
fildes = gopen("example.sbv", OUTDEV, "hpsbv", INIT | SPOOLED);
```

For FORTRAN77 Programs:

```
fildes = gopen('example.sbv'//char(0), OUTDEV, 'hpsbv'//char(0),
               INIT + SPOOLED);
```

For Pascal Programs:

```
fildes = gopen('example.sbv', OUTDEV, 'hpsbv', INIT + SPOOLED);
```

Driver Default

There are a number of driver options that may be manipulated with the Starbase `gescape` function. See the **Gescape** section in this chapter for the defaults and options.

The default action taken without use of the special `gescape` functions results in an archive file containing all the relevant geometry display calls stored as one usable object for the Personal Visualizer.

Starbase Functionality

Commands Supported

The following Starbase commands are supported by the `hpsbv` device driver:

- `move3d` - *(for current point)*
- `arc`
- `ellipse`
- `polygon2d`
- `polygon3d`
- `quadrilateral_mesh`
- `rectangle`
- `triangular_strip`
- `spline_surface`
- `concat_matrix`
- `concat_transformations3d`
- `curve_resolution`
- `flush_matrices`
- `gclose`
- `gerr_control`
- `gescape`
- `gopen`
- `hidden_surface`
- `knot_vectors`
- `make_picture_current`
- `pop_matrix`
- `push_matrix`
- `vertex_format`

Commands Not Supported *(no-ops)*

The Starbase commands not listed above are not supported. They have no effect on the final output, however, some may report errors. These reported errors will not effect the archive file.

Gescapes

The `hpsbv` driver recognizes a number of `gescape` functions. Following are the supported functions and definitions. Details can be found in Appendix A of this manual.

- `SBVESC_OBJ_NAME`—Name the following data by this name.
- `SBVESC_BEGIN_ARC`—Begin archiving all relevant Starbase calls.
- `SBVESC_END_ARC`—Stop archiving any future Starbase calls.
- `SBVESC_COMMENT`—Embed a user defined comment in the archive file (has no effect on output object data).
- `SBVESC_LF_COORD`—Left handed coordinate data.
- `SBVESC_RT_COORD`—Right handed coordinate data.

Gescape Syntax

The `gescape` function allows the application program to input or output to a device in a device dependent manner. The syntax for the `gescape` function is:

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

⋮
gescape_arg arg1, arg2;

⋮
gescape(fildes, ⟨op⟩, &arg1, &arg2);
```

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

The ⟨*op*⟩ is the opcode that specifies the action to be performed. This code is only relevant to a specific driver and is ignored when passed to other devices.

The `arg1` and `arg2` parameters provide additional data needed by a `gescape`.

Troubleshooting

Table 27-1. Troubleshooting Guide

Problem	Probable Solution
Application experiences `gopen` errors.	HPSBV may not have been linked in, or the `SPOOLED` flag was not OR'ed in with the *mode* flag.
The archive file appears unexpectedly small or empty.	Graphics code may be non-polygonal output (such as polylines), or the gescape `SBVESC_END_ARC` was called.
Objects appear flat in the PV and have no depth to them.	Check code to see if data is created by `polygon2d` calls, rather than `polygon3d`, for example. You may also have to open the driver with the `THREE_D` flag.
Object file contains multiple objects I would like to deal with separately.	Try separating the data with 3-D Edit or use the gescape `SBVESC_OBJ_NAME` in your Starbase code.
Objects appear reversed in orientation when imported into the PV.	Rotate them about Y by 180 degrees. The driver attempts to keep positive and negative points relative to their original coordinate systems when converting from left-handed to right-handed space.
Objects appear to project shadows with holes when rendered in the PV.	Use the -b (create double-sided polygons) option in the `sbvtrans` translator. This will create correct polygons for shadow testing in the PV.

Table 27-1. Troubleshooting Guide (continued)

Problem	Probable Solution
The surface features of an object appear to be a mixture of smooth and faceted data.	There can be several reasons for this effect. If the -n (use Starbase normals) option from `sbvtrans` was used, then not all of the data had shading normals per vertex. If total smooth shading is desired, try using the -s option from `sbvtrans`. This will attempt to smooth the object data given the current epsilon setting (-e option). If this operation does not work to your satisfaction, try increasing the epsilon value to consider spaced out points closer in neighborhood. If only selective smoothing is desired, try smoothing the appropriate points through the 3D Edit editor in the PV.
Objects appear to jump off the screen when scaled in the PV.	Your objects require centering when they are first created. Use the centering option (-c) from the `sbvtrans` translator. If you have a group of objects which will remain grouped, use the (-c all) option to center the entire group in the archive.
`sbvtrans` complains about discovering syntax errors in the archive.	Usually this message appears when `sbvtrans` has encounted an unexpected end of file. The archive may have been created without properly closing the file descriptor with a `gclose()` call. The file may also be truncated due to a file system problem. If either of these problems occur, attempt to create a new archive file. (`sbvtrans` will output any geometry created up to the point of the error.)

Table 27-1. Troubleshooting Guide (continued)

Problem	Probable Solution
Object faces appear to "pop" on and off in the PV while manipulating the geometry.	This model has been created with the normals (-n) option of `sbvtrans`. Certain Starbase normals may be incompatible with PV's implementation of vertex normals. Check to see that individual vertex normals are correct through 3-D Edit in the PV. If the faces are wrong, fix them there. If they are correct, try recreating the model without the (-n) option to determine if a model without Starbase normals is correct. If all else fails and a smooth object is desired, try the -s option in `sbvtrans`. (Some popping may occur normally since PV vertex normals are used for backface rejection.) Also, if facets are non-planar, then PV assumptions about planar polygons may cause popping.
Object faces appear to be missing after using the smoothing option (-s) of `sbvtrans`.	Adjoining faces may be creating incompatible vertex normals for smoothing and backface culling. Try 3-D Edit in the PV to fix local cases or adjust vertex points of problem polygons not to be considered exactly adjacent in their Starbase database. Recreate the archive from the original application.
Objects appear white when imported into the PV.	The HPSBV pipeline does not transfer color attributes to the PV. The PV is an attribute driven application, i.e., object elements are separated by color attributes. Try using the `gescape SBVESC_OBJ_NAME` to partition HPSBV output by color. Import the separate objects into the PV and apply appropriate attributes.

Table 27-1. Troubleshooting Guide (continued)

Problem	Probable Solution
Objects appear to be "inside-out".	Check original Starbase code for incorrect use of `gescape SBVESC_RT_COORD`. If this is not the problem, check to see if the correct vertex format statement is being used for clockwise / counter-clockwise definitions. (Turning on hidden_surface culling during normal Starbase displaying will usually reveal the problem from Starbase.)
`sbvtrans` running out of memory.	Try running the translator on another machine where other large applications are not running. It is not advisable to run the translator and other large applications (such as the PV) when extremely large archives are being converted.
PV complains that it cannot run the IMPORT script.	Set correct search path first. Use the command from the VIEW manager: `set path "/users/name/pathname..."` before attempting to run IMPORT scripts.
PV complains that it cannot find objects in the IMPORT script.	The objects may have been relocated from the directories where they were originally created (along with script.) Do not relocate objects to new directories after IMPORT scripts are created since they contain absolute pathnames to all object files at the time of creation.
Objects don't appear in the current viewport after importation.	You may be required to reorient your scene cameras or scaling the geometry to a size which is relevant to the current view extent. Try scaling the objects very small and using the center object option in the View manager of the PV. (Note: This option will only bring the object to the global origin. It will not re-center the data for scaling options described in the trouble shooting hint given in "jump off the screen" ... Try centering your data with the options given in the `sbvtrans` translator.)

Table 27-1. Troubleshooting Guide (continued)

Problem	Probable Solution
Object parts from an archive appear all at the origin.	If the centering option (-c) in `sbvtrans` is invoked, each object data file is centered independent of the rest of the objects in an archive. If an archive consists of an entire group, use the (-c all) option to center all objects as if one group.
Redundant copies of the object appear to be created in the same object.	Check your Starbase code to see that multiple definitions of the object database are not being output to the archive.
Curved surfaces seem to be coarsely faceted.	Try tuning the curve_resolution() function call parameters to improve output for spline surface, arc, and ellipse calls.
Object names imported with the IMPORT script do not match those given in the original gescape calls.	`sbvtrans` attempted to resolve possible name conflicts in an archive before creating output files. If the IMPORT script creates undesirable names, edit the script to create more useful names. These names, however, are limited to five (5) significant characters each. For example: `import wave "/data/john/partfile_a.obj" to partA` becomes `import wave "/data/john/partfile_a.obj" to brake`
I don't remember the `sbvtrans` command I used to translate a particular archive.	Examine the IMPORT file created from translation. It contains a copy of the command line along with the date the archive was translated.

Table 27-1. Troubleshooting Guide (continued)

Problem	Probable Solution
My data contains concave polygons . . . will they render correctly in the PV?	The Personal Visualizer expects planar, convex polygons to render geometry correctly. If a facet is concave and/or twisted to be non-planar, try redefining the face in the PV model editor as a group of triangular facets.
Data imported into the PV does not appear the same way as it did in Starbase (view, clipping, etc . . .)	Data transferred to the PV through the SBV device driver breaks the display transformation pipeline before any view transforms take place. Viewing transforms are applied interactively in the PV, so it is best to imagine the PV as the completion of the entire viewing/clipping pipeline to the SBV driver, i.e., the PV handles the display tasks normally taken care of by Starbase.
My program makes partial polygon calls to create holes, but none show up in the PV.	Partial polygons output data which cannot be properly rendered by the PV. Use the polygon "holes" to redefine new surrounding polygon meshes in the model editor if "holes" are necessary.
The tessellation of my spline surfaces appears uneven.	As a virtual 3-D driver, it makes no sense to use the screen-based flags DC_VALUES, VDC_VALUES, or METRIC for the function curve_resolution() (although they will work). Use the coordinate type STEP_SIZE to create uniform partitioning over an entire surface in modeling coordinates.
How long must I keep SBV archive files and any of the files created by the translator?	After data has been successfully imported into the PV, it is not necessary to retain any of the archive or translator files unless the PV versions need to be recreated.

Table 27-1. Troubleshooting Guide (continued)

Problem	Probable Solution

I've written a system call to SBVTrans from my application and would like to cancel its output to the console.	From the language C, using the command system() requires concatenating the following to your command line.
	`sprintf(cmd, "sbvtrans %s -n -c all", archive);`
	`strcat(cmd, "-q 1> /dev/null 2> /dev/null");`
	`system(cmd);`

28

The Terminal Device Driver

Device Description

The `hpterm` driver supports the following terminals:

- HP 2623A
- HP 2627A
- HP 150A, HP 150-II
- HP 2625A, HP 2628A
- HP 2393A
- HP 2397A

The driver name `hp262x` can also be used with this driver for backward compatibility with earlier releases of Starbase.

Setting Up the Device

Switch Settings

To succeed, proper communication must be established with the terminal before using the terminal driver. The correct settings for the baud rate, parity, etc., must be made. To do this, consult the terminal manuals supplied with your equipment, the *HP-UX System Administrator Manual* and the system administrator for your system.

Note The `IndHndShk(G)` and `Inh DC2(H)` straps are automatically set to `YES` before inquiries are made to establish correct handshaking. The `XmitPace` and `RecvPace` fields in the terminal's datacomm configuration menu should be set by hand to `Xon/Xoff`.

Special Device Files (mknod)

The `mknod` command creates a special device file which is used to communicate between the computer and the terminal. See the `mknod`(1M) information in the *HP-UX Reference* for further details. Since superuser capabilities are needed to create special device files, they are normally created by the system administrator.

Although special device files can be made in any directory of the HP-UX File System, the convention is to create them in the `/dev` directory. Any name may be used for the special device file, however the name that is suggested for these devices is `tty`*xx* for a terminal directly connected to your HP-UX System and `ttyd`*xx* for a remote terminal (dial-up port) connected to your system with a modem.

For the Series 300

When the terminal is a typical hardwired port connection, the `mknod` command should create a character device file with major number 1 and minor number `0x`⟨*sc*⟩⟨*ad*⟩`04`, where ⟨*sc*⟩ is the two-digit select code and ⟨*ad*⟩ is the two-digit port address:

```
mknod /dev/tty26 c 1 0x⟨sc⟩⟨ad⟩04
```

Note that the leading `0x` causes the number to be interpreted hexadecimally.

When the terminal is a dialup modem port, the `mknod` command should create a character device file with major number 1 and minor number `0x`⟨*sc*⟩⟨*ad*⟩`01`, where ⟨*sc*⟩ is the two-digit select code and ⟨*ad*⟩ is the two-digit port address:

```
mknod /dev/ttyd41 c 1 0x⟨sc⟩⟨ad⟩01
```

A `getty`(1M) process must be active on a port before it can be used to log in.

For the Series 800

When the terminal is a typical hardwired port connection, the `mknod` command should create a character device with major number 1 and minor number `0x`⟨*lu*⟩⟨*ad*⟩, where ⟨*lu*⟩ is the two-digit hardware logical unit and ⟨*ad*⟩ is the two-digit mux port address:

```
mknod /dev/tty3p2 c 1 0x00⟨lu⟩⟨ad⟩
```

Note that the leading `0x` causes the number to be interpreted hexadecimally.

A `getty`(1M) process must be active on a port before it can be used to log in.

Note When opening a terminal using `gopen` with the driver as `hp262x` or `hpterm`, the terminal processing for the HP-UX system will temporarily be set for canonical processing. This is done to ensure that the device can respond quickly enough to an inquiry from the driver. Following the inquiry, the previous processing state is restored. The same action is done for the other inquiries during `gopen`.

Linking the Driver

Shared Libraries

The shared driver is the file named `libddhpterm.sl` in the `/usr/lib` directory. The device driver will be explicitly loaded at run time by compiling and linking with the starbase shared library `/usr/lib/libsb.sl`, or by using the `-l` option `-lsb`.

Examples

To compile and link a C program for use with the shared library driver, use:

```
cc example.c -I/usr/include/X11R5/x11 -L/usr/lib/X11R5\
-lXwindow -lsb -lXhp11 -lX11 -ldld -lm -o example
```

or with FORTRAN use,

```
F77 example.f -Wl,-L/usr/lib/X11R5 -lXwindow -lsb\
-lXhp11 -lX11 -ldld -lm  -o example
```

or with Pascal use,

```
pc example.p  -Wl,-L/usr/lib/X11R5 -lXwindow -lsb\
-lXhp11 -lX11 -ldld -lm -o example
```

For details, see the discussion of the `gopen` procedure in the section *To Open and Initialize the Device* in this chapter.

Archive Libraries

The archive driver is located in the `/usr/lib` directory under both file names `libddhpterm.a` and `libdd262x.a`. It may be linked to a program by using the absolute path name `/usr/lib/libddhpterm.a` or `/usr/lib/libdd262x.a`, an appropriate relative path name, or by using one of the `-l` options `-lddhpterm` or `-ldd262x` with the `LDOPTS` environmental variable set to `-a archive`.

The reason for using the `LDOPTS` environmental variable is that the `-l` option will look for a shared library driver first and then look for the archive driver if shared was not found. By exporting the `LDOPTS` variable as specified above, the `-l` option will only look for archive drivers. For more information, refer to the *Programming on HP-UX* manual on linking shared or archive libraries.

Examples

Assuming you are using `ksh`(1), to compile and link a C program for use with this driver, use:

```
export LDOPTS="-a archive"
```

and then:

```
cc example.c -lddhpterm -L/usr/lib/X11R5 -lXwindow\
-lsb1 -lsb2 -lXhp11 -lX11 -lm  -o example
```

or for FORTRAN, use:

```
F77 example.f -lddhpterm -Wl,-L/usr/lib/X11R5 -lXwindow\
 -lsb1 -lsb2 -lXhp11 -lX11 -o example
```

or for Pascal, use:

```
pc example.p -lddhpterm -Wl,-L/usr/lib/X11R5 -lXwindow\
 -lsb1 -lsb2 -lXhp11 -lX11 -o example
```

Device Initialization

Parameters for gopen

The `gopen` call has four parameters: Path, Kind, Driver and Mode.

Path	The name of the special device file created by the **`mknod`** command (for example, `/dev/tty`*xx*.) For the terminal at which you are logged in, the pseudo-device `/dev/tty` can be used and is recommended for programs intended to plot only on the invoker's terminal.
Kind	Parameter which indicates I/O characteristics of the device. This parameter may be one of the following: ■ `OUTDEV`—Output only ■ `INDEV`—Input only ■ `OUTINDEV`—Input and Output
Driver	The functionality of the driver may be specified directly by using a character string that identifies the type of Hewlett-Packard terminal in use, or it may be determined indirectly by allowing the terminal to identify itself. The following strings may be used to specify the terminal type directly: `"hp2623"` `"hp2627"` `"hp150"` `"hp2625"` or `"hp2628"` (functionally equivalent) `"hp2393"` `"hp2397"` The following two strings may be used to indicate that the terminal should identify itself. The strings are functionally equivalent, but should not be used for a spooled output configuration. `"hpterm"` `"hp262x"`

The terminal is expected to respond to a device ID inquiry with a sequence of characters beginning with one of the following:

`2623`
`2627`
`_150` (Terminal ID contains a prepended significant blank)
`2620` (For HP 2625 and HP 2628 Terminals)
`2393`
`2397`
`2390` (For HP 2393 or HP 2397)

Characters appended to these base ID numbers are ignored (for example, `2627A` is acceptable). Terminals with variable device IDs should be configured to an appropriate ID from the above list. If the response from the terminal is not recognized, an error will be generated and the `gopen` call will fail.

If the terminal is configured with the `2390` ID, a color capability inquiry is performed to determine whether the terminal is an HP 2393 or an HP 2397.

Mode The mode control word consisting of several flag bits *or* ed together. Listed below are those flag bits which have device-dependent actions. Those flags not discussed below operate as defined by the `gopen` procedure.

- `0`—open the device, but do nothing else.
- `INIT`—open and initialize the device.
- `SPOOLED`—open the device for spooled operation.

Note Because device inquiries are not possible when output is spooled, the driver type should be selected directly; an error will result and the `gopen` will fail if either `"hpterm"` or `"hp262x"` is specified when `SPOOLED` is also specified.

Syntax Examples

For C Programs:

To open an HP Graphics Terminal:

```
fildes = gopen("/dev/tty", INDEV, "hpterm", INIT);
fildes = gopen("spool_file", OUTDEV, "hp2623", SPOOLED);
fildes = gopen("/dev/tty", OUTDEV, "hp2627", 0);
```

For FORTRAN77 Programs:

To open an HP Graphics Terminal:

```
fildes=gopen('spool_file'//char(0), OUTDEV,
      'hp2393'//char(0), SPOOLED)
```

or

```
fildes=gopen('/dev/tty'//char(0), INDEV,
     'hp2393'//char(0), INIT)
```

or

```
fildes=gopen('/dev/tty'//char(0), OUTDEV,
      'hp2393'//char(0), INIT)
```

For Pascal Programs:

To open an HP Graphics Terminal:

```
fildes := gopen('spool_file', OUTDEV, 'hp2393', SPOOLED);
```

or

```
fildes := gopen('/dev/tty', INDEV, 'hp2393', INIT);
```

or

```
fildes := gopen('/dev/tty', OUTDEV, 'hp2393', INIT);
```

Special Device Characteristics

Screen Resolution

Each of the terminals support a screen resolution of 512×390. Additionally, the HP 2393 and HP 2397 Terminals support a resolution of 640×400 that is selectable in the global config menu. When `SPOOLED` is not specified at `gopen`, a terminal inquiry will be performed for these two terminals to determine its resolution. It is assumed that this resolution will remain unchanged until after `gclose` is called.

The higher resolution can be selected for the HP 2393 and HP 2397 when `SPOOLED` is specified with a `gescape` using ⟨*op*⟩ `HPTERM_640x400`. The two `gescape` arguments are ignored. Since the determination of the screen resolution is normally performed during `gopen` time, the user is required to call `set_p1_p2` with appropriate parameters *immediately after* the call to `gescape` to reset the default transformation matrix.

Polygons

Polygons are generated in software for the HP 2623 and are not limited by the driver in the number of supported vertices. A warning is generated, however, for polygons containing more than 255 vertices.

The driver supports polygon generation for the other terminals in hardware. Because of existing hardware limitations, the driver limits the number of supported vertices. For the HP 2625, HP 2628 and HP 150 terminals the limit is 105 vertices. For the HP 2627, HP 2393 and HP 2397 terminals the limit is 145 vertices. If more vertices are specified than allowed by the limit, the polygon will be truncated and a warning will be generated.

Device Defaults

Default Color Map

The HP 2623, HP 150, HP 2625, HP 2628 and HP 2393 terminals use a monochrome software color map.

The HP 2627 terminal uses three bits to define eight colors in a software color map. (Colors may be changed in the color map with a call to `define_color_table` *before* they are written to the display, but once written remain fixed.) The default color table contains the first eight colors of the standard Starbase Color Map.

Table 28-1. Default Color Table

Pen	Color	Red	Green	Blue
0	black	0.0	0.0	0.0
1	white	1.0	1.0	1.0
2	red	1.0	0.0	0.0
3	yellow	1.0	1.0	0.0
4	green	0.0	1.0	0.0
5	cyan	0.0	1.0	1.0
6	blue	0.0	0.0	1.0
7	magenta	1.0	0.0	1.0

The HP 2397 Terminal also has an 8-color color table (or, "palette 0") that is initialized to the standard Starbase Color Map when `INIT` is specified in the `gopen` procedure. The pre-existing color map is used when `INIT` is not specified. The map, however, uses six bits per color, allowing each of the 8 colors in the color map to be set to one of 64 possible values. Because the color map is implemented in hardware, previously written colors may change with calls to `define_color_table`.

The RGB colors passed to `define_color_table` are rounded according to the color resolution of the terminal. Colors for monochromatic terminals are rounded to black or white, colors for the HP 2627 are rounded to the closest of 8 possible values and colors for the HP 2397 are rounded to the closest of 64 possible values. The rounding will be reflected in the RGB values returned by `inquire_color_table`.

Dither Default

Dithering is supported in hardware by two color terminals, HP 2627 and HP 2397, with dithering mode off by default. Selecting the number of dither colors to be 2, 4, 8 or 16 selects the terminal's hardware dithering capability to be on when direct-filling polygons with RGB fill colors. Dithering is turned off by setting the number of dither colors to 1 using the `fill_dither()` procedure.

Dithering on the HP 2397 Terminal assumes that the hardware color map contains power-on color assignments. Unfortunately, these do not correspond to the standard Starbase Color Map (assigned to the HP 2397 when `INIT` is specified at `gopen` time). To make dithering results accurate on the HP 2397, the color map needs to be assigned the following values:

Table 28-2. HP 2397 Power-up Color Table

Pen	Color	Red	Green	Blue
0	black	0.0	0.0	0.0
1	red	1.0	0.0	0.0
2	green	0.0	1.0	0.0
3	yellow	1.0	1.0	0.0
4	blue	0.0	0.0	1.0
5	magenta	1.0	0.0	1.0
6	cyan	0.0	1.0	1.0
7	white	1.0	1.0	1.0

Note that color map assignments are not important when dithering on an HP 2627 since its hardware pen assignments are always fixed (the color map is in software and dithering is in hardware). It is recommended that this difference between the HP 2627 and the HP 2397 be accounted for, however, when using both dithered fills and indexed color selections in applications intended for both terminals.

Line Types

Line types are defined in *Starbase Reference* under `line_type`(3g).

- The default line type is line type 0, i.e. `solid`.
- This device driver defines line type 7 to be the same as line type 4, and 6 to be the same as 3.
- This device driver defines line type -1 as terminal line type 11 (point plotting), and line type -2 the same as terminal line type 9.

Starbase Functionality

Commands Not Supported (no-ops)

The following commands are not supported. An error will *not* be generated if any of these commands are called.

```
alpha_transparency
await_retrace
backface_control
background_color
background_color_index
bank_switch
bf_alpha_transparency
bf_control
bf_fill_color
bf_interior_style
bf_perimeter_color
bf_perimeter_repeat_length
bf_perimeter_type
bf_surface_coefficients
bf_surface_model
bf_texture_index
bitmap_print
bitmap_to_file
block_move
block_read
block_write
clear_control
contour_enable
dbuffer_switch
dcbitmap_print
dcbitmap_to_file
dcblock_move
dcblock_read
dcblock_write
define_contour_table
define_raster_echo
define_texture
define_trimming_curve
deformation_mode
depth_cue
depth_cue_color
depth_cue_range
depth_cue-range
display_enable
double_buffer
drawing_mode
file_to_bitmap
file_to_dcbitmap
file_to_intbitmap
fill_dither
hidden_surface
intbitmap_print
intbitmap_to_file
intblock_move
intblock_read
intblock_write
intline_width
light_ambient
light_attenuation
light_model
light_source
light_switch
line_endpoint
line_filter
pattern_define
perimeter_filter
set_capping_planes
set_model_clip_indicator
set_model_clip_volume
shade_mode
shade_range
surface_coefficients
surface_model
texture_index
texture_viewport
texture_window
viewpoint
write_enable
zbuffer_switch
```

Conditionally Supported

The following commands are supported under the listed conditions:

`define_color_table`	Except for the HP 2397, only the software color map may be defined.
`echo_type`	Some types are defaulted.
`fill_dither`	Supported only on the HP 2627 and the HP 2397.
`interior_style`	Only the `INT_SOLID`, `INT_HOLLOW`, and `INT_HATCH` styles are supported.
`line_type`	Some line types are approximated.
`set_locator`	Hewlett-Packard Terminals do not support independent echo and locator positions. Therefore, in order to preserve the echo's position, the set locator call sets only the Z coordinate of the locator's position.
`vertex_format`	The ⟨use⟩ parameter must be zero. Any extra coordinates will be ignored.

`with_data`	`partial_polygon_with_data3d`
	`polygon_with_data3d`
	`polyhedron_with_data`
	`polyline_with_data3d`
	`polymarker_with_data3d`
	`quadrilateral_mesh_with_data`
	`triangle_strip_with-data`
	Additional data per vertex will be ignored if not supported by this device. For example, contouring data will be ignored if the device does not support it.

Text

Hardware-generated text may be selected by setting the text precision to STRING_TEXT. One of eight possible character sizes may be selected by specifying an approximate height or width. The results returned by inquire_text_extent will be affected by the character slant but are not affected by special characters such as a tab, carriage return or line feed. Text alignment default is device dependent. To alter the alignment for STRING_TEXT, use the gescape functions HPTERM_PRINT_ESC or HP26_PRINT_ESC to send the control string to the device. Your terminal reference manual contains the details of the control strings for altering device dependent alignment.

Raster Operations

This device driver does not support block_read, block_write and block_move. Starbase calls to perform these operations are treated as no-ops.

Terminal Device Access

Note that only one program can access the terminal driver at a time or the terminal will get confused. Also note the program can only gopen the terminal once or the terminal will again get confused.

Input

Tracking from a terminal is not supported. Continuously sampling a terminal in a loop without significant delay can exceed the terminal's ability to execute commands; therefore, the terminal should not be continuously sampled. Sampling during a request or while events are enabled may cause a keypress to be missed. Therefore, sampling while requesting or while events are enabled is discouraged.

Although requesting events (e.g. key presses) from a terminal is allowed, it is strongly discouraged to do so while any graphics operations may be occurring. Because the event queue is discarded prior to graphics rendering, events may be lost, and the results may be unpredictable.

The same terminal status request is used for device requests or for locator requests. This causes the graphics cursor to appear while in choice request mode. The keyboard driver and the terminal driver cannot be used simultaneously for input from the same device because they interfere with each other's operation.

Echo for the HP 2623

Echo types 0 and 3 are supported. Echo types specified as other values are mapped into echo type 3.

Echo for Other Terminals

Echo types 0, 3 and 4 are supported. Echo types specified as other values are mapped into echo type 3.

The terminal graphics cursor is not visible while the terminal is plotting lines. Consequently, a rapid loop that alternates between drawing and updating the echo position may cause the cursor to flicker or disappear altogether.

Drawing Mode

The driver approximates Starbase drawing modes with those supported by the terminals. See your terminal's reference manual for further details. Monochromatic terminals support five drawing modes, `NOP`, `CLEAR`, `SET`, `COMPLEMENT` and `JAM`. Color terminals support eight drawing modes, `NOP`, `CLEAR1`, `JAM1`, `COMP1`, `JAM2`, `OR`, `COMP2` and `CLEAR2`. The following table shows the mapping from Starbase drawing modes to terminal drawing modes.

Table 28-3. Drawing Mode Replacement Rule

Starbase Replacement Rule for drawing_mode Command	Monochromatic Replacement Rule		Color Replacement Rule	
	Number	**Mnemonic**	**Number**	**Mnemonic**
0	1	CLEAR	1	CLEAR1
1	4	JAM	7	CLEAR2
2	4	JAM	7	CLEAR2
3 (default)	2	SET	2	JAM1
4	4	JAM	7	CLEAR2
5	0	NOP	0	NOP
6	3	COMPLEMENT	6	COMP2
7	2	SET	5	OR
8	4	JAM	7	CLEAR2
9	3	COMPLEMENT	6	COMP2
10	3	COMPLEMENT	3	COMP1
11	2	SET	5	OR
12	1	CLEAR	1	CLEAR1
13	2	SET	5	OR
14	4	JAM	7	CLEAR2
15	4	JAM	4	JAM2

If the Starbase drawing mode is changed from the default (3) value for monochromatic terminals, no color attributes changes will be recognized. You must be in drawing mode 3 to change color attributes, e.g., `line_color`, `fill_color`, etc.

When the drawing mode is set to a complement mode, a condition may exist where line end-points are drawn twice, resulting in some endpoints being complemented twice. This condition can occur when performing a non-line block operation (for example, setting an attribute) between successive move/draw/polyline operations.

Parameters for gescape

The `hpterm` driver supports the following `gescapes`. A detailed discussion on `gescape` functions can be found in Appendix A of this manual.

The `READ_COLOR_MAP` `gescape` is common to two or more devices. The following `gescape` functions are unique for this driver:

- `HPTERM_640x400`—Set high-resolution spooled output
- `HPTERM_PRINT_ESC` or `HP26_PRINT_ESC`—Send terminal control (escape) strings
- `READ_COLOR_MAP`—Copy the hardware color map to software color map

Gescapes

Introduction

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

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 A-1. Supported Operation Codes (op)

⟨*op*⟩ **Parameter**	**Function**
`AUTO_RESIZE_DEVICE`	Automatically scale graphics output when the window size changes.
`BAD_SAMPLE_ON_DIFF_SCREEN`	Restores the locator and choice sampling of the X11 pointer device.
`BLINK_INDEX`	Alternate Between Hardware Color Maps.
`BLINK_PLANES`	Blink the display using a mask.
`BLOCK_WRITE_SKIPCOUNT`	Specifies number of bytes to be skipped at the end of each scanline.
`CGMESC_ENCODING`	Selects CGM encoding.
`CGMESC_MET_NAME`	Defines metafile name.
`CGMESC_TOP_MODE`	Selects TOP mode for metafile generation.
`CGMESC_VDC_PREC`	Selects VDC integer precision.
`CGMESC_APPL_DATA`	Generates CGM application data element.
`CGMESC_ESCAPE_ELT`	Generates CGM escape element.
`CGMESC_FONT_IX`	Allows application to select fonts.
`CGMESC_MESSAGE`	Generates CGM message element.
`CGMESC_PIC_NAME`	Defines picture name.
`CLIP_OVERFLOW`	Change X Window system hierarchy.
`CONTOUR_CONTROL`	Specifies alternative methods for interpolation of contour data.
`CUBIC_POLYPOINT`	specify points to be rendered in a cubic volume specified in modeling coordinates.
`DC_COMPATIBILITY_MODE`	Controls rendering of DC polygons.

Table A-1. Supported Operation Codes (op) (continued)

⟨*op*⟩ **Parameter**	**Function**
DC_PIXEL_WRITE	Specify points to be rendered along a horizontal scan line.
DISABLE_ACKNOWLEDGE	Disables bell function.
DISABLE_AUTO_PROMPT	Disable HP-HIL auto prompt.
DRAW_POINTS	Select different modes of rounding for rendered points.
ENABLE_ACKNOWLEDGE	Allows bell character when request/event is satisfied.
ENABLE_AUTO_PROMPT	Enable HP-HIL auto prompt.
GAMMA_CORRECTION	Enable/disable gamma correction.
GCRX_PIXEL_REPLICATE	Allows you to pan and zoom a raster image.
GCRX_SW_CMAP_FULL	Redefines default behavior.
GR2D_CONVEX_POLYGONS	Enables convex polygons to be drawn at a higher speed.
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	Sets multiple plane bit/pixel mask.
GR2D_PLANE_RULE	Specifies rules per plane for frame buffer bit/pixel block writes.
GR2D_REPLICATE	Allows square pixel replication.

Table A-1. Supported Operation Codes (op) (continued)

⟨*op*⟩ **Parameter**	**Function**
`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.
`HPGL2_ADAPTIVE_LINES`	Determines adaptive or fixed line types.
`HPGL2_CUTTER_CONTROL`	Enable/disable paper cutter.
`HPGL2_FONT_POSTURE`	Indicates upright or italic font posture.
`HPGL2_FONT_TYPEFACE`	Selects typeface.
`HPGL2_FONT_WEIGHT`	Sets the font stroke weight independent of Starbase.
`HPGL2_LOGICAL_PEN_WIDTH`	Determines the logical pen width.
`HPGL2_REPLOT`	Indicates number of replots for the command buffer.
`HPGL2_SET_CMAP_SIZE`	Indicates the size of the color map: number of pens available.
`HPGL2_SET_MEDIA_TYPE`	Determines the type of media to be used.
`HPGL2_SET_QUALITY`	Indicates the quality level of the output.
`HPTERM_640x400`	Set high-resolution spooled output
`HPTERM_PRINT_ESC` or `HP26_PRINT_ESC`	Send terminal control (escape) strings
`IGNORE_PROXIMITY`	Ignores stylus proximity.

Table A-1. Supported Operation Codes (op) (continued)

⟨*op*⟩ **Parameter**	**Function**
`IGNORE_RELEASE`	Trigger when button pressed.
`ILLUMINATION_ENABLE`	Specify amount of illumination data per vertex.
`IMAGE_BLEND`	Enable/disable video blending.
`INQ_12_BIT_INDEXING`	Indicates if display mode is 12 bit indexing.
`LINEAR_POLYPOINT`	Specify points to be rendered along a line specified in modeling coordinates.
`LS_OVERFLOW_CONTROL`	Sets options for light source overflow situations.
`OLD_SAMPLE_ON_DIFF_SCREEN`	Inquires the locator and choice sampling of the X11 pointer device.
`OVERLAY_BLEND`	Control blending of overlay plane frame buffer.
`PAN`	Pixel pan only.
`PAN_AND_ZOOM`	Pixel pan and zoom.
`PATTERN_FILL`	Fills polygon with stored pattern.
`PLUG_ACCELERATED_PIPELINE`	Controls the rendering of the graphics accelerators into the frame buffer.
`POLYGON_TRANSPARENCY`	Define front facing and backfacing polygon transparency patterns.
`PROMPT_OFF`	Switch prompt indicator off.
`PROMPT_ON`	Switch prompt indicator on.
`R_BIT_MASK`	Identifies the plane(s) to read to or write from.
`R_BIT_MODE`	Changes the `raw` mode flag.

Table A-1. Supported Operation Codes (op) (continued)

$\langle op \rangle$ **Parameter**	**Function**
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_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.
R_TRANSPARENCY_INDEX	Specify transparency index.
R_UNLOCK_DEVICE	Unlocks the specified device.
READ_COLOR_MAP	Reads the color map.
REPORT_PROXIMITY	Reports stylus proximity.
SBVESC_BEGIN_ARC	Begin archiving all relevant Starbase calls.

Table A-1. Supported Operation Codes (op) (continued)

⟨*op*⟩ Parameter	Function
SBVESC_COMMENT	Embed a user defined comment in the archive file (has no effect on output object data).
SBVESC_END_ARC	Stop archiving any future Starbase calls.
SBVESC_LF_COORD	Left handed coordinate data.
SBVESC_OBJ_NAME	Name the following data by this name.
SBVESC_RT_COORD	Right handed coordinate data.
SET_ACCELERATION	Set acceleration and threshold values.
SET_BANK_CMAP	Install frame buffer bank color maps.
SET_REPLACEMENT_RULE	Set replacement rules for bit/pixel writes.
SMD_ALLOCATE_MEMORY	Allocate frame buffer
SMD_DEFINE_DEPTH	Define memory buffer depth
SMD_DEFINE_XY	Define X, Y dimensions
SMD_GET_MEM_REQUIRED	Determining memory requirements
SMD_SUPPLY_MEM_BUFF	Supply memory buffer
STEREO	Supports stereoscopic display systems.
SWITCH_SEMAPHORE	Controls the device access semaphores.
TC_HALF_PIXEL	Allows access to half pixels.
TEXTURE_CONTROL	Selects texture map filter.
TEXTURE_DOWNSAMPLE	Downsamples defined texture into off screen
TEXTURE_RETRIEVE	Retrieves downsampled texture from off screen.
TOGGLE_2D_COLORMAP	Enables/disables 2D colormap.
TRANSPARENCY	Allows "screen door" for transparency pattern.

Table A-1. Supported Operation Codes (op) (continued)

⟨*op*⟩ Parameter	Function
`TRIGGER_ON_RELEASE`	Trigger when button released.
`ZBUFFER_ALLOC`	Allocates frame buffer memory for Starbase.
`ZSTATE_RESTORE`	Allows creation of 3D cursors in overlay.
`ZSTATE_SAVE`	Allows creation of 3D cursors in overlay.
`ZWRITE_ENABLE`	Allows creation of 3D cursors in overlay.

Table A-2. Supported Device Drivers

⟨*op*⟩ Parameter	Supported Drivers
AUTO_RESIZE_DEVICE	hp98736, hp98766
BAD_SAMPLE_ON_DIFF_SCREEN	hp98550, hp98556, hp98704, hp98705, hp98720, hp98721, hp98730, hp98731, hp98735, hp98736, hp98765, hp98766, hpcrx48z
BLINK_INDEX	hp98704, hp98705, hp98720, hp98721 , hp98730, hp98731, hp98735, hp98736, hp98765, hp98766
BLINK_PLANES	(blink speed 2.4 Hz) hp300l, hp300h, hp98704, hp98705, hp98720, hp98721, hp98550, hp98556, hp98730, hp98731, hp98735, hp98736 hp98765, hp98766
BLOCK_WRITE_SKIPCOUNT	hp98735, hp98736, hp98765, hp98766, hpevrx, hpgcrx, hpcrx48z
CGMESC_ENCODING	hpcgm
CGMESC_MET_NAME	hpcgm
CGMESC_TOP_MODE	hpcgm
CGMESC_VDC_PREC	hpcgm
CGMESC_APPL_DATA	hpcgm
CGMESC_ESCAPE_ELT	hpcgm
CGMESC_FONT_IX	hpcgm
CGMESC_MESSAGE	hpcgm
CGMESC_PIC_NAME	hpcgm
CLIP_OVERFLOW	hp98731
CONTOUR_CONTROL	hp98736, hp98766, hpcrx48z
CUBIC_POLYPOINT	hpgcrx (CRX-24/CRX-24Z only), hpcrx48z
DC_PIXEL_WRITE	hpgcrx (CRX-24/CRX-24Z only), hpcrx48z

Table A-2. Supported Device Drivers (continued)

⟨*op*⟩ Parameter	Supported Drivers
DC_COMPATIBILITY_MODE	hp98736, hp98766
DISABLE_ACKNOWLEDGE	lkbd
DISABLE_AUTO_PROMPT	hphil
DRAW_POINTS	hp98736, hp98766, hpcrx48z
ENABLE_ACKNOWLEDGE	lkbd
ENABLE_AUTO_PROMPT	hphil
GAMMA_CORRECTION	hp98705,hp98731, hp98735, hp98736,hp98765, hp98766, hpgcrx (CRX-24 and CRX-24Z only), hpcrx48z
GCRX_PIXEL_REPLICATE	hpgcrx, hpcrx48z
GCRX_SW_CMAP_FULL	hpgcrx
GR2D_CONVEX_POLYGONS	hp98556
GR2D_DEF_MASK	hp98550, hp98556, hp98735, hp98736, hp98765, hp98766
GR2D_FILL_PATTERN	hp98550, hp98556, hp98735, hp98736
GR2D_MASK_ENABLE	hp98550, hp98556, hp98735, hp98736, hp98765, hp98766
GR2D_MASK_RULE	hp98550, hp98556, hp98735, hp98736, hp98765, hp98766
GR2D_OVERLAY_TRANSPARENT	hp98550, hp98556, hp98735, hp98736, hp98765, hp98766
GR2D_PLANE_MASK	hp98550, hp98556, hp98704, hp98705, hp98735, hp98736, hp98765, hp98766
GR2D_PLANE_RULE	hp98735, hp98736, hp98765, hp98766

Table A-2. Supported Device Drivers (continued)

⟨*op*⟩ Parameter	Supported Drivers
GR2D_REPLICATE	hp98550, hp98556, hp98735, hp98736, hp98765, hp98766
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
HPGL2_ADAPTIVE_LINES	CADplt2
HPGL2_CUTTER_CONTROL	CADplt2
HPGL2_FONT_POSTURE	CADplt2
HPGL2_FONT_TYPEFACE	CADplt2
HPGL2_FONT_WEIGHT	CADplt2
HPGL2_LOGICAL_PEN_WIDTH	CADplt2
HPGL2_REPLOT	CADplt2
HPGL2_SET_CMAP_SIZE	CADplt2
HPGL2_SET_MEDIA_TYPE	CADplt2
HPGL2_SET_QUALITY	CADplt2
HPTERM_640x400	hpterm
HPTERM_PRINT_ESC or HP26_PRINT_ESC	hpterm
IGNORE_PROXIMITY	hphil

Table A-2. Supported Device Drivers (continued)

⟨*op*⟩ **Parameter**	**Supported Drivers**
IGNORE_RELEASE	hphil, X11, hpevrx, hpgcrx, hpcrx48z
ILLUMINATION_ENABLE	hp98736, hp98766, hpgcrx with Powershade, hpcrx48z
IMAGE_BLEND	hp98730, hp98731
INQ_12_BIT_INDEXING	hp98735, hp98736, hp98765, hp98766
LINEAR_POLYPOINT	hpgcrx (CRX-24/CRX-24Z only), hpcrx48z
LS_OVERFLOW_CONTROL	hp98705, hp98721, hp98731, hp98736, hp98766, hpgcrx with Powershade, hpcrx48z
OLD_SAMPLE_ON_DIFF_SCREEN	hp98550, hp98556, hp98704, hp98705, hp98720, hp98721, hp98730, hp98731, hp98735, hp98736, hp98765, hp98766, hpcrx48z
OVERLAY_BLEND	hp98730, hp98731
PAN	hp98735, hp98736, hp98765, hp98766
PAN_AND_ZOOM	hp98730, hp98731
PATTERN_FILL	hp98705, hp98721, hp98731, hp98736, hp98766
PLUG_ACCELERATED_PIPELINE	hp98735, hp98736, hpcrx48z
POLYGON_TRANSPARENCY	hp98731, hp98736, hp98766, hpgcrx with Powershade, hpcrx48z
PROMPT_OFF	hphil
PROMPT_ON	hphil
R_BIT_MASK	hp300l, hp300h, hp98704, hp98705, hp98720, hp98721, hp98550, hp98556, h98730, hp98731, hp98735, hp98736, hp98765, hp98766, hpevrx, hpgcrx, hpcrx48z

Table A-2. Supported Device Drivers (continued)

⟨*op*⟩ Parameter	Supported Drivers
R_BIT_MODE	hp300l, hp300h, hp98704, hp98705, hp98720, hp98721, hp98550, hp98556, hp98730, hp98731, hp98735, hp98736, hp98765, hp98766, hpevrx, hpgcrx, hpcrx48z
R_DEF_ECHO_TRANS	hp98704, hp98705, hp98720, hp98721, hp98550, hp98556, hp98730, hp98731, hp98735, hp98736, hp98765, hp98766, hpcrx48z
R_DEF_FILL_PAT	hp300l, hp300h, hp98704, hp98705, hp98720, hp98721, hp98550, hp98730, hp98731, hp98735, hp98736, hp98765, hp98766
R_DMA_MODE	hp98730, hp98731 Models 825 and 835 SPUs with an A10474 interface card
R_ECHO_CONTROL	hp98704, hp98705,hp98730, hp98731, hp98735, hp98736, hp98765, hp98766
R_ECHO_FG_BG_COLORS	hp98704, hp98705, hp98730, hp98731, hp98735, hp98736, hp98765, hp98766, hpcrx48z
R_ECHO_MASK	hp98704, hp98705, hp98730, hp98731, hp98735, hp98736, hp98765, hp98766, hpcrx48z
R_FULL_FRAME_BUFFER	hp300l, hp300h, hp9836a, hp98704, hp98705, hp98720, hp98721, hp98550, hp98556, hp98730, hp98731, hp98735, hp98736, hp98765, hp98766
R_GET_FRAME_BUFFER	hp300l, hp300h, hp9836a, hp98704, hp98705, hp98720, hp98721, hp98550, hp98556, hp98730, hp98731, hp98735, hp98736, hp98765, hp98766, hpevrx, hpgcrx, hpcrx48z
R_LINE_TYPE	hp98704, hp98705, hp98720, hp98721, hp98730, hp98731, SMD, hp98735, hp98736, hp98765, hp98766, hpevrx, hpgcrx, hpcrx48z

Table A-2. Supported Device Drivers (continued)

⟨*op*⟩ **Parameter**	**Supported Drivers**
R_LOCK_DEVICE	hp3001, hp300h, hp98704, hp98705, hp98720, hp98721, hp98550, hp98556, hp98730, hp98731, hp98735, hp98736, hp98765, hp98766, hpevrx, hpgcrx, hpcrx48z
R_OFFSCREEN_ALLOC	hp98550, hp98556, hp98704, hp98705, hp98730, hp98731, hp98735, hp98736, hp98765, hp98766
R_OFFSCREEN_FREE	hp98550, hp98556, hp98704, hp98705, hp98730, hp98731, hp98735, hp98736, hp98765, hp98766
R_OV_ECHO_COLORS	hp98704, hp98705, hp98720, hp98721, hp98730, hp98731 hp98736, hp98736, hp98765, hp98766
R_OVERLAY_ECHO	hp98704, hp98704, hp98720, hp98721, hp98550, hp98556, hp98730, hp98735, hp98736, hp98765
R_TRANSPARENCY_INDEX	hp98704, hp98705, hp98720, hp98721, hp98730, hp98731, hp98735, hp98736, hp98765, hp98766
R_UNLOCK_DEVICE	hp3001, hp300h, hp98704, hp98705, hp98720, hp98721, hp98550, hp98556, hp98730, hp98731, hp98735, hp98736, hp98765, hp98766, hpevrx, hpgcrx, hpcrx48z
READ_COLOR_MAP	hpterm, hp3001, hp300h, hp98704, hp98705, hp98720, hp98721, hp98550, hp98556, hp98730, hp98731, hp98735, hp98736, hp98765, hp98766, hpevrx, hpgcrx, hpcrx48z
REPORT_PROXIMITY	hphil
SBVESC_BEGIN_ARC	hpsbv
SBVESC_COMMENT	hpsbv

Table A-2. Supported Device Drivers (continued)

⟨*op*⟩ Parameter	Supported Drivers
SBVESC_END_ARC	hpsbv
SBVESC_LF_COORD	hpsbv
SBVESC_OBJ_NAME	hpsbv
SBVESC_RT_COORD	hpsbv
SET_ACCELERATION	hphil
SET_BANK_CMAP	hp98730, hp98731, hp98735, hp98736
SET_REPLACEMENT_RULE	hp98704, hp98705
SMD_ALLOCATE_MEMORY	memory
SMD_DEFINE_DEPTH	memory
SMD_DEFINE_XY	memory
SMD_GET_MEM_REQUIRED	memory
SMD_SUPPLY_MEM_BUFF	memory
STEREO	hp98735, hp98736, hp98765, hp98766, hpgcrx(CRX-24/CRX-24Z only), hpcrx48z
SWITCH_SEMAPHORE	hp300l, hp300h, hp9836a, hp98704, hp98705, hp98720, hp98721, hp98550, hp98556, hp98730, hp98731, hp98735, hp98736, hp98765, hp98766, hpevrx, hpgcrx, hpcrx48z
TC_HALF_PIXEL	hp300l
TEXTURE_CONTROL	hp98736, hp98766
TEXTURE_DOWNSAMPLE	hp98736, hp98766
TEXTURE_RETRIEVE	hp98736, hp98766

Table A-2. Supported Device Drivers (continued)

⟨*op*⟩ Parameter	Supported Drivers
TOGGLE_2D_COLORMAP	hp98735, hp98736, hp98766
TRANSPARENCY	hp98705, hp98721, hp98731, hp98736, hp98766, hpgcrx with Powershade
TRIGGER_ON_RELEASE	hp-hil, X11, hpevrx, hpgcrx
ZBUFFER_ALLOC	hp98721
ZSTATE_RESTOR	hp98721
ZSTATE_SAVE	hp98721
ZWRITE_ENABLE	hp98705, hp98721, hp98731, hp98736, hp98766

AUTO_RESIZE_DEVICE

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

This `gescape` will enable or disable automatic device resizing whenever the window size is changed. Whenever this `gescape` is enabled and the window size is changed, all subsequent graphics primitives will be scaled to the new window size.

An equivalent way to enable `AUTO_RESIZE_DEVICE` is to set the environment variable `SB_AUTO_RESIZE_DEVICE` to a non-null value before starting the graphics program.

The following attributes are recomputed whenever the `AUTO_RESIZE_DEVICE` `gescape` is enabled and the window is resized:

- P1/P2 is reset using the parameters last specified in the last call to `set_p1_p2`.
- Picking window limits are reset for the new device size.
- Line repeat length is reset for the new device size.
- Polygon perimeter line repeat length is reset for the new device size.
- Non-DC marker size is reset for the new device size.

The following attributes are NOT recomputed when the window size is changed:

- Attributes that are set using DC coordinates, including `dcmarker_size`, `dccharacter_height`, and `dccharacter_width`.
- The values specified by `curve_resolution` and `hatch_spacing`.
- Starbase cursors and tracking are NOT affected by a window resize. Cursor and tracking device limits remain the same after a window resize.

This `gescape` has no effect on non-window devices.

C Syntax Example

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

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

FORTRAN77 Syntax Example

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

Pascal Syntax Example

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

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

BAD_SAMPLE_ON_DIFF_SCREEN

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

This gescape restores the locator and choice sampling of the X11 pointer device.

The X server's locator position can be sampled anytime, and is returned relative to the window. By default, when the X pointer is on a different screen than the window, the valid parameter of the sample_locator procedure is returned as FALSE.

When the gescape `OLD_SAMPLE_ON_DIFF_SCREEN` is used, the valid parameter is returned as TRUE when the X pointer is on a different screen. In this case, the pointer position returned by sample_locator is either the last value of the X locator on that screen or the value (0,0) if the pointer has never been on that screen.

To restore the default (valid set to FALSE) behavior, use the `BAD_SAMPLE_ON_DIFF_SCREEN` gescape.

BLINK_INDEX

The ⟨*op*⟩ parameter is BLINK_INDEX.

Refer to the table, **Supported Device Drivers**, at the front of this chapter for devices that support this gescape.

The accelerated devices (devices with transform engines) have two separate hardware color maps. They alternate between the two color maps every 133 ms (milliseconds). 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 you 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 affect 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 details.

To blink color map planes see the gescape for BLINK_PLANES.

C Syntax Example

```
/* 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 Example

```
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 Example

```
{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 `BLINK_INDEX` segment of those device driver sections that allow color map blinking.

The following drivers support a blink speed of 2.4 Hertz:

```
hp3001   hp300h   hp98704   hp98705
```

The following drivers support a blink speed of 3.75 Hertz:

```
hp98720  hp98721  hp98550  hp98556  hp98730  hp98731

hp98735  hp98736
```

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.

BLOCK_WRITE_SKIPCOUNT

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

This `gescape` specifies the number of bytes to be skipped in the source data at the end of each scanline during byte/pixel block writes. The default value is 0. In order for the skipcount to take effect, the **raw** parameter must be set in the call to block_write. This gescape has no effect if bit per pixel mode has been enabled via the `R_BIT_MODE` gescape.

This `gescape` should be called with `arg1.i[0]` containing the number of bytes to be skipped.

C Syntax Example

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

gescape_arg arg1, arg2;
      ⋮
arg1.i[0]=32;
gescape(fildes,BLOCK_WRITE_SKIPCOUNT,&arg1,&arg2);
```

FORTRAN77 Syntax Example

```
integer*4 arg1(64),arg2(64)
arg1(1)=32
call gescape(fildes,BLOCK_WRITE_SKIPCOUNT,arg1,arg2)
```

Pascal Syntax Example

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

var
   arg1,arg2:gescape_arg;
        ⋮
begin
   arg1.i[1] := 32;
   gescape(fildes,BLOCK_WRITE_SKIPCOUNT,arg1,arg2);
```

CGMESC_APPL_DATA

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

This `gescape` generates CGM application data element.

CGM has an element that has no graphical effect at all but can be used to insert documentation or other private data into the metafile. With a clear text metafile generation, for example, you can use this `gescape` to insert comments into the metafile (as debugging aids). This clarifies the correspondence between high level Starbase calls and clear text CGM elements.

The CGM application data element has two parameters: an application data ID and a data record. The ID is a label for the application data element. The data record contains parameters.

The `arg1` parameter contains:

an integer application data ID
one or more blanks
a data record substring (commencing with the first non-blank character)

The `arg2` parameter is ignored.

C Syntax Example

```
gescape_arg  arg2;
    ⋮
gescape(fildes, CGMESC_APPL_DATA,
    "10   APPLICATION move/draw", &arg2);
```

FORTRAN77 Syntax Example

```
 character arg2(255)
     ⋮
 call gescape(fildes, CGMESC_APPL_DATA,
+  '10   APPLICATION move/draw'//char(0), arg2)
```

Pascal Syntax Example

```
var arg1, arg2: gescape_arg;
    :
arg1.c := '10   APPLICATION move/draw'#0;
gescape(fildes, CGMESC_APPL_DATA,arg1, arg2);
```

CGMESC_ENCODING

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

This `gescape` selects CGM encoding.

CGMs may be encoded in one of three style: binary, character, or clear text (see "Functionality and Encodings" earlier in this chapter).

The `arg1` parameter contains one of the one-letter strings "`B`", "`C`", or "`T`" to select binary, character, or clear text encodings, respectively.

The `arg2` parameter is ignored.

The default encoding is binary.

C Syntax Example

A

```
gescape_arg  arg2;
     ⋮
gescape(fildes, CGMESC_ENCODING, "T", &arg2);
```

FORTRAN Syntax Example

```
character arg2(255)
   ⋮  call gescape(fildes, CGMESC_ENCODING, 'T'//char(0), arg2)
```

Pascal Syntax Example

```
var arg1, arg2: gescape_arg;
     ⋮
arg1.c[1] := 'T';
gescape(fildes, CGMESC_ENCODING, arg1, arg2);
```

CGMESC_ESCAPE_ELT

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

This `gescape` generates a CGM escape element.

The CGM contains an escape element that defines non-standardized graphical operations. For example, one could define an escape element that suppressed the clearing of the view surface when the metafile is interpreted; hence, pictures would be overlaid. (This example is a **registered escape** of the TOP application profile standard.)

Because the CGM escape contents are inherently non-standard, portability of the resulting metafiles is inherently reduced by using this element.

The CGM escape element has two parameters: an escape ID and an escape data record. The ID is an opcode, and the data record contains parameters.

The `arg1` parameter contains:

an integer opcode.
one or more blanks
an escape data record substring (commencing with the first non-blank character)

The `arg2` parameter is ignored.

C Syntax Example

```
gescape_arg  arg2;
   ⋮
gescape(fildes, CGMESC_ESCAPE_ELT,"-302  1.0 0.0 0.0 0.22" , &arg2);
```

FORTRAN Syntax Example

```
      character arg2(255)
          ⋮
      call gescape(fildes, CGMESC_ESCAPE_ELT,
     +  '-302  1.0 0.0 0.0 0.22'//char(0), arg2)
```

Pascal Syntax Example

```
var arg1, arg2: gescape_arg;
      ⋮
arg1.c := '-302  1.0 0.0 0.0 0.22'#0;
gescape(fildes, CGMESC_ESCAPE_ELT,arg1, arg2);
```

CGMESC_FONT_IX

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

The CGM contains an element to set the current font index. This is an index into the interpreter font table which allows selection of the font to be used for subsequent text display. There is no way to directly define this font index in Starbase. Hence, this `gescape` allows the application to select different fonts in the metafile.

The `arg1` parameter contains the integer index encoded as a string.

The `arg2` parameter is ignored.

The default font index is 1.

C Syntax Example

```
gescape_arg  arg2;
    ⋮
gescape(fildes, CGMESC_FONT_IX, "12", &arg2);
```

FORTRAN Syntax Example

```
character arg2(255)
    ⋮
call gescape(fildes, CGMESC_FONT_IX, '12'//char(0), arg2)
```

Pascal Syntax Example

```
var arg1, arg2: gescape_arg;
    ⋮
arg1.c := '12'#0;
gescape(fildes, CGMESC_FONT_IX,arg1, arg2);
```

A

CGMESC_MESSAGE

The ⟨*op*⟩ parameter is `CGMESC_MESSAGE`. This `gescape` generates a CGM message element. The CGM contains an element to pass a message to an operator at the other end, i.e., at the interpretation process. Such a message might inform the operator that a certain kind of paper is required in the plotter for the next pictures. This `gescape` allows the application to generate a CGM message element.

The `arg1` parameter contains the string that comprises the message.

Note The CGM element has an action flag as a parameter. This `gescape` always generates message elements with the value `no_action` for this flag.

A

The `arg2` parameter is ignored.

C Syntax Example

```
gescape_arg  arg2;
    ⋮
gescape(fildes, CGMESC_MESSAGE,"Next is the move/draw polygon", &arg2);
```

FORTRAN Syntax Example

```
      character arg2(255)
          ⋮
      call gescape(fildes, CGMESC_MESSAGE,
     +  'Next is the move/draw polygon'//char(0), arg2)
```

Pascal Syntax Example

```
var arg1, arg2: gescape_arg;
    ⋮
arg1.c := 'Next is the move/draw polygon'#0;
gescape(fildes, CGMESC_MESSAGE,arg1, arg2);
```

CGMESC_MET_NAME

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

This `gescape` defines a metafile name.

Each CGM begins with an element `BEGIN METAFILE` having an ID string as a parameter. This `gescape` defines the name that appears in the metafile ID string.

The `arg1` parameter contains the ID string that is used.

The `arg2` parameter is ignored.

The default value is the null string.

C Syntax Example

```
gescape_arg  arg2;
    ⋮
gescape(fildes,CGMESC_MET_NAME, "HP-CGM metafile name",&arg2);
```

FORTRAN Syntax Example

```
 character arg2(255)
     ⋮
 call gescape(fildes, CGMESC_MET_NAME,
+  'HP-CGM metafile name'//char(0), arg2)
```

Pascal Syntax Example

```
var arg1, arg2: gescape_arg;
    ⋮
arg1.c := 'HP-CGM metafile name'#0;
gescape(fildes,CGMESC_MET_NAME,arg1,arg2);
```

CGMESC_PIC_NAME

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

This `gescape` defines the picture name.

In the CGM each picture begins with a `BEGIN PICTURE` element that contains an ID string to name the picture. This `gescape` defines the name that appears in the picture ID string for the next picture to be started in the metafile.

The `arg1` parameter contains the picture ID string that is used.

The `arg2` parameter is ignored.

The default value is the null string.

C Syntax Example

A

```
gescape_arg  arg2;
    ⋮
gescape(fildes, CGMESC_PIC_NAME, "Picture name", &arg2);
```

FORTRAN Syntax Example

```
 character arg2(255)
     ⋮
 call gescape(fildes, CGMESC_PIC_NAME,
+  'Picture name'//char(0), arg2)
```

Pascal Syntax Example

```
var arg1, arg2: gescape_arg;
    ⋮
arg1.c := 'Picture name'#0;
gescape(fildes, CGMESC_PIC_NAME, arg1, arg2);
```

CGMESC_TOP_MODE

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

This `gescape` selects the TOP mode for metafile generation.

The resulting metafile will conform to the MAP/TOP V3.0 Application Profile (AP) of CGM. This is a specification which limits the ranges of attributes to a predictable set, changes the default color map, and limits the lengths of primitives to 1024 points. The purpose of the AP is to promote predictable interchange of CGM by removing some ambiguities that exist in the CGM standard itself.

The default color map, starting at index 2, is redefined to red, green, blue, yellow, magenta, cyan, black, and white. This pattern of colors is repeated until the entire 256-element color map is filled. Indexes 0 and 1 are not redefined; hence, they are black and white.

There are no parameters for this `gescape`.

The `arg1` and `arg2` parameters are ignored.

The default mode is non-TOP.

C Syntax Example

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

FORTRAN Syntax Example

```
character arg1(255), arg2(255)
    ⋮
call gescape(fildes, CGMESC_TOP_MODE, arg1, arg2)
```

Pascal Syntax Example

```
var arg1, arg2: gescape_arg;
    ⋮
gescape(fildes, CGMESC_TOP_MODE, arg1, arg2);
```

CGMESC_VDC_PREC

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

This `gescape` selects the VDC integer precision.

Coordinate data in a CGM may be either high or low precision (see "Precisions").

The parameter `arg1` contains one of the one-letter strings "`H`" or "`L`" to select high or low precision coordinates for graphical primitives and attributes. In low precision, coordinates range from zero to +32,767. In high precision, coordinates range from zero to +1,000,000,000.

The parameter `arg2` is ignored.

The default precision is low.

C Syntax Example

```
gescape_arg  arg2;
   ⋮
gescape(fildes, CGMESC_VDC_PREC, "L", &arg2);
```

FORTRAN Syntax Example

```
character arg2(255)
   ⋮
call gescape(fildes, CGMESC_VDC_PREC, 'L'//char(0), arg2)
```

Pascal Syntax Example

```
var arg1, arg2: gescape_arg;
   ⋮  arg1.c[1] := 'L';
gescape(fildes, CGMESC_VDC_PREC, arg1, arg2);
```

CGM Elements Produced by the HP CGM Driver

Delimiter Elements

Every CGM created by `hpcgm` contains the following delimeter elements.

BEGIN METAFILE	The metafile ID can be specified by a `gescape`. (⟨*op*⟩ parameter set to `CGMESC_MET_NAME`)
BEGIN PICTURE	The picture ID can be specified by a `gescape`. (⟨*op*⟩ parameter set to `CGMESC_PIC_NAME`)
BEGIN PICTURE BODY	
END METAFILE	
END PICTURE	

Metafile Descriptor, Picture Descriptor, Control Elements

Unconditionally Included

The following Metafile Descriptor, Picture Descriptor, and Control Elements are included in all `hpcgm` metafiles. The descriptions of some of the precisions refer to the encoding-dependent nature of the parameters (binary, character, or clear text).

BACKGROUND COLOR	Value according to most recent application request to Starbase, or 0, 0, 0 if no requests have been made.
CHARACTER CODING ANNOUNCER	Always `basic 7-bit`.
COLOR INDEX PRECISION	Always 8-bit (or closest equivalent supported by the selected encoding).
COLOR PRECISION	Always 8-bit (or closest equivalent supported by the selected encoding).
COLOR SELECTION MODE	Always "`direct`".
COLOR VALUE EXTENT	Always (0,0,0), (255,255,255).
INDEX PRECISION	Always 16-bit (or closest equivalent supported by the selected encoding).
INTEGER PRECISION	Always 16-bit (or closest equivalent supported by the selected encoding).
LINE WIDTH SPECIFICATION MODE	Always "`scaled`".
MARKER SIZE SPECIFICATION MODE	Always "`scaled`".

MAXIMUM COLOR INDEX	Always 255.
METAFILE DEFAULTS REPLACEMENT	Unconditionally sets the proper VDC integer precision (to 16-bit or 32-bit, or closest equivalent supported by the encoding).
METAFILE DESCRIPTION	Always contains the substring `Hewlett-Packard CGM   (HP-CGM) 1987`.
METAFILE ELEMENT LIST	Contains drawing set.
METAFILE VERSION	Version fixed at 1.
REAL PRECISION	Always 32-bit (or closest equivalent supported by the selected encoding). Fixed point is used.
SCALING MODE	Always "`abstract`"
VDC TYPE	Always an integer.

Unconditionally Excluded

The following descriptor and control elements never appear in a `hpcgm` metafile.

```
AUXILIARY COLOR
CHARACTER SET LIST
CLIP INDICATOR
CLIP RECTANGLE
EDGE WIDTH SPECIFICATION MODE
FONT LIST
TRANSPARENCY
VDC EXTENT
VDC INTEGER PRECISION
VDC REAL PRECISION
```

Graphical Primitives

Included

The following CGM graphical primitives may be generated as a result of user Starbase calls. Polylines may reflect such things as stroke precision text which will be simulated by Starbase.

```
POLYGON
POLYGON SET
POLYLINE
TEXT
```

Excluded

The following CGM graphical primitives will never be generated by the `hpcgm` driver.

```
APPEND TEXT
CELL ARRAY
CIRCLE
CIRCULAR ARC CENTRE
CIRCULAR ARC CENTRE CLOSE
CIRCULAR ARC 3 POINT
CIRCULAR ARC 3 POINT CLOSE
DISJOINT POLYLINE
ELLIPSE
ELLIPTICAL ARC
ELLIPTICAL ARC CLOSE
POLYMARKER
RECTANGLE
RESTRICTED TEXT
```

Primitive Attributes

Included

The hpcgm driver may put the following CGM primitive attribute elements into a metafile as a result of application calls to Starbase functions.

CHARACTER EXPANSION FACTOR	
CHARACTER HEIGHT	
CHARACTER ORIENTATION	
CHARACTER SPACING	
FILL COLOR	
INTERIOR STYLE	Hollow or solid may be output.
LINE COLOR	
LINE TYPE	Starbase 0 ... 4 are mapped to CGM 1 ... 5. Starbase values greater than 4 are mapped to CGM -(value+1).
TEXT ALIGNMENT	Continuous alignment is always used.
TEXT COLOR	
TEXT FONT INDEX	May be included as a result of gescape. (The *op* parameter is set to CGMESC_FONT_IX.)
TEXT PATH	

Excluded

The following CGM primitive attribute elements will never be output by the hpcgm driver.

```
ALTERNATE CHARACTER SET INDEX
ASPECT SOURCE FLAGS
CHARACTER SET INDEX
COLOR TABLE
EDGE BUNDLE INDEX
EDGE COLOR
EDGE TYPE
EDGE VISIBILITY
EDGE WIDTH
FILL BUNDLE INDEX
FILL REFERENCE POINT
```

HATCH INDEX
LINE BUNDLE INDEX
LINE WIDTH
MARKER BUNDLE INDEX
MARKER COLOR
MARKER SIZE
MARKER TYPE
PATTERN INDEX
PATTERN SIZE
PATTERN TABLE
TEXT BUNDLE INDEX
TEXT PRECISION

External and Escape Elements

The CGM external and escape elements may be output by `hpcgm` by `gescape` calls to Starbase.

APPLICATION DATA	(⟨*op*⟩ parameter is set to `CGMESC_APPL_DATA`)
ESCAPE	(⟨*op*⟩ parameter is set to `CGMESC_ESCAPE_ELT`)
MESSAGE	Only available with ⟨*noaction*⟩. (⟨*op*⟩ parameter is set to `CGMESC_MESSAGE`)

CLIP_OVERFLOW

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

This `gescape` allows you to provide the hp98731 driver a routine to change the X Window system window hierarchy when the window that the hp98731 driver is using becomes too obscured by other windows. It takes a single parameter, which is the address of the routine to call.

The `arg1` parameter points to the address.

The `arg2` parameter is ignored.

The hp98730 transform engine has the ability to clip against a limited number of obscuring rectangles. When too many rectangles obscure a window, by default, the hp98731 driver prints a Starbase warning and waits for the situation to change. It will continue to print warnings until the number of obscuring rectangles is fewer than 31.

With the `CLIP_OVERFLOW` `gescape`, it is possible for you to provide the driver a routine to call instead of printing the warning. This will allow the application to fix the problem immediately. To put the driver back in the default state (printing warnings), call `CLIP_OVERFLOW` with a null address.

When calling the user routine, the hp98731 driver will pass in two parameters: the display the window is on, and the window the driver is writing to. It is possible to use these parameters in X Window system calls. The user routine should not call any Starbase routines.

Here is an example of how to use the `CLIP_OVERFLOW` `gescape`:

C Syntax Example

```
void fixit(display,window)
Display *display;
Window  window;
{
/* This routine will try to raise the window to the top if possible. */

XRaiseWindow(display,window);

}

main()
{
int fildes;
gescape_arg arg1,arg2;

fildes = gopen(...,OUTDEV,"hp98731",0);
arg1.i[0] = (int) fixit;
gescape(fildes,CLIP_OVERFLOW,&arg1,&arg2);

Do drawing

gclose(fildes);
}
```

FORTRAN77 and Pascal Syntax

Since FORTRAN77 and Pascal cannot get the address of a procedure, this `gescape` does not directly support those languages.

CONTOUR_CONTROL

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

This `gescape` specifies alternative methods for interpolating the scalar data provided for contoured primitives. This `gescape` takes 3 floating point numbers in `arg1` to set the option. Two global contouring values can be specified through `arg1.f[1]` and `arg1.f[2]`. These floating point numbers are treated respectively as the real and imaginary parts of a complex number. The default value for both of these numbers is 1.0. The alternative modes treat each of the two per vertex contour scalar values as complex numbers as well, with the first contour value being the real part and the second contour value being the imaginary part. With these alternative modes, it is possible couple deformation and contouring such that contoured deformed primitives can vary the contours as the deformation varies. The contouring mode is selected through `arg1.f[0]` and can be either:

`SCALAR_FRONT_BACK`	This is the default option. Front facing contours are computed as the product of the value of `arg1.f[1]` and the first contour value per vertex. Back facing contours are computed as the product of the value of `arg1.f[2]` and the second contour value per vertex. To get this option the `gescape` should be called with: `arg1.f[0]=SCALAR_FRONT_BACK`
`SCALAR_REAL`	This option contours with the real part of the complex product of the global and per vertex contour values. To get this option the gescape should be called with: `arg1.f[0]=SCALAR_REAL`
`SCALAR_IMAG`	This option contours with the imaginary part of the complex product of the global and per vertex contour values. To get this option the gescape should be called with: `arg1.f[0]=SCALAR_IMAG`

The following examples select a `SCALAR_REAL` contouring interpolation method using a non default global contouring complex number.

C Syntax Example

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

```
gescape_arg arg1, arg2;
   :
arg1.f[0] = SCALAR_REAL;
arg1.f[1] = 2.0;              /* global contouring number real component */
arg1.f[2] = 5.5;              /* global contouring number imaginary component */
gescape(fildes,CONTOUR_CONTROL,&arg1,&arg2);
```

FORTRAN77 Syntax Example

```
real arg1(64),arg2(64)
arg1(1) = SCALAR_REAL
arg1(2) = 2.0         /* global contouring number real component */
arg1(3) = 5.5         /* global contouring number imaginary component */
call gescape(fildes,CONTOUR_CONTROL,arg1,arg2)
```

Pascal Syntax Example

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

var
   arg1,arg2:gescape_arg;
       :
begin
   arg1.f[1] := 1.0;
   arg1.f[1] := SCALAR_REAL;
   arg1.f[2] := 2.0;          /* global contouring number real component */
   arg1.f[3]) := 5.5;         /* global contouring number imaginary component */
   gescape(fildes,CONTOUR_CONTROL,arg1,arg2);
```

CUBIC_POLYPOINT

The CUBIC_POLYPOINT gescape provides a means to specify points to be rendered in a cubic volume specified in modeling coordinates. For each point specified by a data value, one pixel will be rendered (see AUTO_FILL_VOXEL below).

The data may be provided in any orientation relative to the device coordinate (DC) space. This entrypoint sorts the data so as to always render the points from back to front in the DC Z axis. The data specified is composited with the current contents of the framebuffer.

The compositing function is of the form:

```
        new value = (alpha * source) + ( (1 - alpha) * destination)
```

where source is the supplied value and destination is the current frame buffer contents. Compositing is performed separately for the red, green, and blue banks. This primitive assumes a 24-bit CMAP_FULL shade-mode (or 12/12 CMAP_FULL double-buffer).

This gescape does not support perspective transformations.

The arg2 parameter is ignored.

Input Parameters:

"ORIGIN: Start of the 1st scanline in the 1st slice"

arg1.f[0]	X value in Modeling coordinates
arg1.f[1]	Y value in Modeling coordinates
arg1.f[2]	Z value in Modeling coordinates

"End of the 1st scanline in the 1st slice"

arg1.f[3]	X value in Modeling coordinates
arg1.f[4]	Y value in Modeling coordinates
arg1.f[5]	Z value in Modeling coordinates

"Start of the last scanline in the 1st slice"

arg1.f[6]	X value in Modeling coordinates
arg1.f[7]	Y value in Modeling coordinates

`arg1.f[8]`	Z value in Modeling coordinates

"Start of the 1st scanline in the last slice"

`arg1.f[9]`	X value in Modeling coordinates
`arg1.f[10]`	Y value in Modeling coordinates
`arg1.f[11]`	Z value in Modeling coordinates
`arg1.i[12]`	Number of points per scanline to render
`arg1.i[13]`	Number of scanlines per slice to render
`arg1.i[14]`	Number of slices to render
`arg1.i[15]`	Number of voxels (data values) per point to skip
`arg1.i[16]`	Number of points per scanline to skip (at the end of each line)
`arg1.i[17]`	Number of lines per slice to skip at end of each slice
`arg1.i[18]`	Vertex format flags
`arg1.i[19]`	Pointer to the data values (long words, packed bytes, or packed shorts)
`arg1.i[20]`	Pointer to the data mapping table scalar(byte) -> alpha,r,g,b (this parameter applies only to the INDIRECT vertex formats)

The allowed values for the vertex format flag are:

`NULL`	Standard format is a pack word per voxel. Each word contains an byte of: (alpha, red, green, blue) with each byte scaled in the range of 0-255.
`INDIRECT_DATA`	Indicates that the vertex format is a packed byte array of monotonic intensities which map into a 256 entry (alpha, red, green, blue) lookup table supplied via a pointer in arg1.i[20].
`INDIRECT_DATA16`	Indicates that the vertex format is a packed short (16 bit) array of intensity which maps into a 65,536 entry (alpha, red, green, blue) lookup table supplied via a pointer in arg1.i[20].

`PRE_SCALED_ALPHA`	Indicates that the vertex format is per the CRX-24/24Z's fast path (1-alpha, alpha*r, alpha*g, alpha*b). The CRX-24 will operate most efficiently (quickly) if the direct data or indirect table is pre-scaled as described here. This pre-scaling will avoid having the gescape scale the data per voxel (for the DIRECT case) or per gescape (for the INDIRECT case). Note that the CRX-48Z can directly handle alpha, red, green, and blue. For the CRX-48Z this option is slower and should not be used, although it will still work.
`AUTO_FILL_VOXEL`	This flag enables a filling algorithm which will render a rectangle for each voxel. The rectangle encloses the voxel cube projection and allows the DC image to be enlarged without rarifying the appearance of the rendered volume. If this flag is not present, a single pixel is modified per data point, regardless of the DC spacing between data points.

DC_COMPATIBILITY_MODE

The $\langle op \rangle$ parameter is `DC_COMPATIBILITY_MODE`.

The DC_COMPATIBILITY_MODE gescape controls the rendering of DC polygons when using the HP 98736 device driver. By default, the HP 98736 device driver will draw dc polygons inclusive of both the top and bottom scanlines in the definition. It does so at the expense of speed and will, in the case of some partial dc polygons, draw some scanlines twice. Invoking this `gescape` will cause the top scanline of all dc polygons to not be drawn, however dc polygon speed will increase and no scanlines of the polygon will be drawn more than once. This mode is recommended for applications which require dc polygon speed or which are using dc polygons in anti-aliasing, destination dependent drawing modes, alpha blending, or any sort of compositing.

This `gescape` is needed only with the HP 98736 device driver.

Setting `arg1.i[0]=0` will cause dc polygons to be drawn without their top scanline and will increase their speed. Setting `arg1.i[0]=1` will cause dc polygons to be drawn completely, sometimes drawing scanlines more than once.

C Syntax Example

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

FORTRAN77 Syntax Example

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

Pascal Syntax Example

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

var
   arg1,arg2:gescape_arg;

begin
   arg1.i[1] := 0;
   gescape(fildes,DC_COMPATIBILITY_MODE,arg1,arg2);
```

DC_PIXEL_WRITE

The DC_PIXEL_WRITE gescape provides the means to specify points to be rendered along a horizontal scan line. Successive pixels along the scan line are rendered in a left to right manner.

The data specified is composited with the current contents of the frame buffer.

The compositing function is of the form:

```
new value = (alpha * source) + ( (1 - alpha) * destination)
```

where source is the supplied value and destination is the current frame buffer contents. Compositing is performed separately for the red, green and blue banks. This primitive assumes a 24-bit CMAP_FULL shade-mode (or 12/12 CMAP_FULL double-buffer).

The arg2 parameter is ignored.

Input Parameters:

arg1.i[0]	X starting value in device coordinates
arg1.i[1]	Y starting value in device coordinates
arg1.i[2]	Number of points to render
arg1.i[3]	Vertex format flags
arg1.i[4]	pointer to the data values (long words, packed bytes, or packed shorts)
arg1.i[5]	pointer to the data mapping table scalar(byte) -> alpha,r,g,b (this parameter applies only to the INDIRECT vertex formats)
	The allowed values for the vertex format flag are:
NULL	Standard format is a pack word per voxel. Each word contains an byte of: (alpha, red, green, blue) with each byte scaled in the range of 0-255.
INDIRECT_DATA	indicates that the vertex format is a packed byte array of monotonic intensity which maps into a 256 entry (alpha, red, green, blue) lookup table supplied via a pointer in arg1.i[20].

`INDIRECT_DATA16`	indicates that the vertex format is a packed short (16 bit) array of intensity which maps into a 65,536 entry (alpha, red, green, blue) lookup table supplied via a pointer in arg1.i[20].
`PRE_SCALED_ALPHA`	indicates that the vertex format is per the CRX-24/24Z's fast path (1-alpha, alpha*r, alpha*g, alpha*b). The CRX-24 will operate most efficiently (quickly) if the direct data or indirect table is pre-scaled as described here. This pre-scaling will avoid having the gescape scale the data per voxel (for the DIRECT case) or per gescape (for the INDIRECT case). Note that the CRX-48Z can directly handle alpha, red, green, and blue. For the CRX-48Z this option is slower and should not be used, although it will still work.

DISABLE_ACKNOWLEDGE

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

This `gescape` disables the acknowledge function described under `gescape` `ENABLE_ACKNOWLEDGE`.

The default condition is acknowledge disabled.

`arg1` and `arg2` are ignored.

C Syntax Example

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

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

FORTRAN77 Syntax Example

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

Pascal Syntax Example

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

var
   arg1,arg2:gescape_arg;
begin
   gescape(fildes,DISABLE_ACKNOWLEDGE,arg1, arg2);
```

DISABLE_AUTO_PROMPT

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

This `gescape` disables the auto prompt facility enabled by the previously discussed procedure. The prompt indicator will not be activated automatically after this `gescape` is executed. You can manually turn the prompt indicator on and off with the `PROMPT_ON` and `PROMPT_OFF` escape codes described next.

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,DISABLE_AUTO_PROMPT,&arg1,&arg2);
```

FORTRAN77 Syntax Example

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

Pascal Syntax Example

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

DRAW_POINTS

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

This `gescape` allows you to select different modes of rounding for rendered points. A move and draw to the same point or a polyline to the same point will not necessarily appear on HP 98736 nor CRX-24Z as it did on previous devices. This is because HP 98736 and CRX-24Z have subpixel resolution and only those points that end up on pixel centers are drawn. This `gescape` is used to provide point drawing compatibility with images rendered on previous devices. The `gescape` takes 1 integer number in `arg1` to set the type of rounding that will occur. These options are:

`NATIVE MODE`	This is the native mode for HP 98736 and CRX-24Z. No rounding of points will occur. To get this option the `gescape` should be called with: `arg1.i[0]=0`
`CENTER VECTORS`	This mode forces all vector primitives to round x,y positions to pixel center locations. This mode should be chosen to provide compatibility with previous devices for vector primitives. To get this option the gescape should be called with: `arg1.i[0]=1`
`CENTER TEXT`	This is the default mode for the HP 98736 and the only mode for the CRX-24Z. It forces only rendered text to round x,y positions to pixel center locations. To get this option the gescape should be called with: `arg1.i[0]=2`

One can chose to have both `CENTER VECTORS` and `CENTER TEXT` by setting `arg1.i[0]=3`.

The following examples select `CENTER VECTORS` and `CENTER TEXT`.

C Syntax Example

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

gescape_arg arg1, arg2;
      ⋮
arg1.i[0]=3;
gescape(fildes,DRAW_POINTS,&arg1,&arg2);
```

FORTRAN77 Syntax Example

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

Pascal Syntax Example

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

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

ENABLE_ACKNOWLEDGE

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

Most keyboards have associated with them a tone generator (bell) that can be used to indicate to the operator that an input has been received. This `gescape` causes the driver to write a bell character (7) to the device whenever a request or event is satisfied.

The default condition is acknowledge disabled.

`arg1` and `arg2` are ignored.

C Syntax Example

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

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

FORTRAN77 Syntax Example

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

Pascal Syntax Example

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

var
   arg1,arg2:gescape_arg;
begin
   gescape(fildes,ENABLE_ACKNOWLEDGE,arg1, arg2);
```

ENABLE_AUTO_PROMPT

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

Some HP-HIL devices have an indicator to inform the operator that the device is being accessed. This `gescape` enables this indicator whenever a request starts or events are enabled. If a specific device does not have such an indicator, this procedure is ignored. 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,ENABLE_AUTO_PROMPT,&arg1,&arg2);
```

FORTRAN77 Syntax Example

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

Pascale Syntax Example

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

GAMMA_CORRECTION

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

This `gescape` allows you to enable or disable gamma correction in the HP 98730, 98731, 98735, 98736, 98765, and 98766 hardware.

See the end of this section for information on how gamma correction works on the `hpgcrx` CRX-24 and CRX-24Z and the `hpcrx48Z` CRX-48Z.

You pass in a flag which is set to 1 to enable gamma correction, or 0 to disable gamma correction.

The `arg1` parameter points to the flag.

The `arg2` parameter is ignored.

When enabled with this `gescape`, gamma correction will be performed in the hardware on subsequent primitives rendered in `CMAP_FULL` mode (see `shade_mode`) when using the following display modes:

- 8-planes single buffered with dithering (three bits red, three bits green, three bits blue)
- 16-planes double buffered (eight planes per buffer) with dithering (three bits red, three bits green, two bits blue)
- 24-planes single buffered (eight bits red, eight bits green, eight bits blue)
- 24-planes double buffered (12 planes per buffer) with dithering (four bits red, four bits green, four bits blue)

If the color map or display modes are not in the above set, primitives will be rendered as normal. If the modes are later changed into one of the above cases, gamma correction will be automatically engaged. Therefore, it is possible to enable gamma correction with one call to this `gescape` and switch in and out of modes which will use it.

The gamma correction hardware is actually a pre-computed, look-up table which accepts 10-bit intensity inputs for each color from the scan conversion hardware and outputs 8-bit gamma corrected values. This means that the actual values written to the frame buffer are modified. The color map is unchanged, so previously rendered primitives are unaffected. Also, raster operations such as `block_write` are unaffected by gamma correction.

gamma correction has no effect on performance.

The following example shows how to use Starbase to enter and exit the gamma correction mode.

C Syntax Example

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

 gescape_arg arg1, arg2;

 arg1.i[0] = 1; /* enable gamma correction */
 gescape(fildes,GAMMA_CORRECTION,&arg1,&arg2);
    ⋮
Render gamma corrected primtives here.  Be sure to set the
correct color map and display modes (see shade_mode,
double_buffer, and fill_dither.)
   ⋮
 arg1.i[0] = 0; /* disable gamma correction */
 gescape(fildes,GAMMA_CORRECTION,&arg1,&arg2);
```

FORTRAN77 Syntax Example

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

arg1(1)=1
call gescape(fildes,GAMMA_CORRECTION,arg1,arg2)
   ⋮
Render gamma corrected primtives here.  Be sure to set the
correct color map and display modes (see shade_mode,
double_buffer, and fill_dither.)
   ⋮
arg1(1)=0
call gescape(fildes,GAMMA_CORRECTION,arg1,arg2)
```

Pascal Syntax Example

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

var
   arg1,arg2:gescape_arg;

begin
   arg1.i[1] := 1;
   gescape(fildes,GAMMA_CORRECTION,arg1,arg2);
       ⋮
  Render gamma corrected primtives here.  Be sure to set the
  correct color map and display modes (see shade_mode,
  double_buffer, and fill_dither.)
      ⋮
   arg1.i[1] := 0;
   gescape(fildes,GAMMA_CORRECTION,arg1,arg2);
```

GAMMA_CORRECTION for the CRX-24, CRX-24Z, and CRX-48Z

On the CRX-24, CRX-24Z and CRX-48Z displays, gamma correction is implemented differently. It is done by modifying the `CMAP_FULL` color map. It is not done during rendering, as was the case on the above devices. This means that when gamma correction turned on, everything rendered up to this point will be gamma corrected.

When enabled with this gescape, gamma correction on the CRX-24 and the CRX-24Z works only in the following `CMAP_FULL` display modes:

- 24 planes single buffered
- 24 planes double buffered (12 planes per buffer)

Since this technique of gamma correction is different, a different gescape value is used to turn it on:

`arg1[0] = 0`	Turn off gamma correction
`arg1[0] = 2`	Turn on gamma correction in color map (on CRX-24, CRX-24Z or CRX-48Z)

`arg1[0] = 1`	Does nothing on CRX-24, CRX-24Z or CRX-48Z

If you want to turn on gamma correction regardless of what device you are on, using whichever technique the device supports, use the value:

`arg1[0] = 3`	Turn on gamma correction on any device that supports it.

GCRX_PIXEL_REPLICATE

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

This gescape allows you to pan and zoom a raster image on any of the `hpgcrx` and `hpcrx48z` displays. It works on both a raw display and in an X11 window. It does not support backing store. You pass in a location to write the image to, the size of the image, the zoom factor, and a pointer to the image. The values of these parameters are similar to dcblock_write.

`arg1.i[0] =`	zoom factor (must be integer >=1).
`arg1.i[1] =`	x location in dc units to write the upper left corner.
`arg1.i[2] =`	y location in dc units to write the upper left corner.
`arg1.i[3] =`	width of the raster image in pixels.
`arg1.i[4] =`	height of the raster image in pixels.
`arg1.i[5] =`	pointer to a byte per pixel raster image.

C Syntax

```
gescape_arg arg1 arg2;
unsigned char pixel_data[width*height];

arg1.i[0]=5;            /* zoom factor */
arg1.i[1]=0;            /* x */
arg1.i[2]=0;            /* y */
arg1.i[3]=width;        /* width */
arg1.i[4]=height;       /* height */
arg1.i[5]=pixel_data;   /* image */

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

FORTRAN77 Syntax

```
integer*4 arg1(6), arg2(6)
integer*1 pixel_data(width*height)

arg1(1)=5
arg1(2)=0
arg1(3)=0
arg1(4)=width
arg1(5)=height
arg1(6)=pixel_data

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

Pascal Syntax

```
var
   arg1, arg2 : gescape_arg;
   pixel_data : array [1..width*height] of char;

arg1.i[1]:=5;           { zoom factor }
arg1.i[2]:=0;           { x }
arg1.i[3]:=0;           { y }
arg1.i[4]:=width;       { width }
arg1.i[5]:=height;      { height }
arg1.i[6]:=pixel_data;  { image }

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

When using a zoom factor > 1, the frame buffer area to be written must contain zeros for this gescape to work properly. This can be done either implicitly by doing a dbuffer_switch call when double buffering, or explicitly by using the clear_view_surface routine.

Example:

```
gescape_arg arg1 arg2;
unsigned char pixel_data[width*height];

background_color_index(fildes,0);
```

```
dbuffer_switch(fildes, 0);

arg1.i[0]=5;             /* zoom factor */
arg1.i[1]=0;             /* x */
arg1.i[2]=0;             /* y */
arg1.i[3]=width;         /* width */
arg1.i[4]=height;        /* height */
arg1.i[5]=pixel_data;    /* image */

gescape(fildes,GCRX_PIXEL_REPLICATE,&arg1,&arg2);
dbuffer_switch(fildes, 1);
```

This gescape does not do any CMAP_FULL translation when the modified colormap is in use. The pixel data to be written must be untranslated (i.e. exactly how it appears in the frame buffer). This is different from block_read and block_write calls which normally do use translation when necessary. The modified colormap (and CMAP_FULL translation) is used on the GRX, CRX, and the Dual CRX displays in CMAP_FULL mode (see the *CMAP_FULL Translations* section in the hpgcrx chapter of the *Starbase Device Driver's Manual*).

To obtain the untranslated data from the frame buffer, turn translation off before doing block_read by using the GCRX_SW_CMAP_FULL gescape (see example below). If you are creating your data some other way, use the translation tables in the hpgcrx chapter of the *Starbase Device Drivers Manual* (See the CMAP_FULL Colormap Index Translation section).

Example:

```
gescape_arg arg1 arg2;
unsigned char pixel_data[width*height];

/* The GCRX_SW_CMAP_FULL gescape is used to turn off
   translation.  It is only needed when modified
   colormap is used.
 */
arg1.i[0] = 0;
gescape(fildes,GCRX_SW_CMAP_FULL,&arg1,&arg2);

/* Read the data from the frame buffer without translation. */
dcblock_read(fildes,0,0,width,height,pixel_data,FALSE);
```

```
/* Clear frame buffer to zeros. Translation is off for this also.*/
background_color_index(fildes,0);
clear_view_surface(fildes);

/* Write data back with a zoom factor.  No translation is used. */
arg1.i[0]=5;            /* zoom factor */
arg1.i[1]=0;            /* x */
arg1.i[2]=0;            /* y */
arg1.i[3]=width;        /* width */
arg1.i[4]=height;       /* height */
arg1.i[5]=pixel_data;   /* image */
gescape(fildes,GCRX_PIXEL_REPLICATE,&arg1,&arg2);
```

GCRX_SW_CMAP_FULL

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

This `gescape` allows you to redefine default behavior of the `SW_CMAP_FULL` translation for `CMAP_FULL` color map mode on `hpgcrx` devices.

You pass in a value where 0 means to disable the `SW_CMAP_FULL` translation, 1 means to enable the `SW_CMAP_FULL` translation without changing the enabled value for clearing of the display surface when switching in and out of `CMAP_FULL` color map mode, and 3 means to enable the `SW_CMAP_FULL` translation and the clearing of the display surface when switching in and out of `CMAP_FULL` color map mode.

The `arg1[0]` parameter contains the value as described above.

The `arg2` parameter is ignored.

C Syntax Example

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

gescape_arg arg1, arg2;

arg1.i[0] = 3;
gescape(fildes,GCRX_SW_CMAP_FULL,&arg1,&arg2);
```

FORTRAN77 Syntax Example

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

arg1(1)=3
call gescape(fildes,GCRX_SW_CMAP_FULL,arg1,arg2)
```

Pascal Syntax Example

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

var
   arg1,arg2:gescape_arg;

begin
   arg1.i[1] := 3;
   gescape(fildes,GCRX_SW_CMAP_FULL,arg1,arg2);
```

GR2D_CONVEX_POLYGONS

The ⟨*op*⟩ parameter is `GR2D_CONVEX_POLYGONS`.

This `gescape` enables convex polygons to be drawn at a higher speed than they would normally be with the `gescape` not enabled. This extra speed is achieved at the expense of not being able to draw non-convex polygons when this `gescape` is enabled. If an application attempts to render non-convex polygons while this `gescape` is enabled, they will be filled incorrectly.

The default mode is that the `convex polygons` mode is not enabled.

The `arg1` parameter enables (if `TRUE (1)`) and disables (if `FALSE (0)`) the `convex polygon` mode.

The `arg2` parameter is ignored.

The following examples enable the `convex polygons` mode.

C Syntax Example

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

gescape_arg arg1, arg2;

arg1.i[0]=TRUE;
gescape(fildes,GR2D_CONVEX_POLYGONS,&arg1,&arg2);
```

FORTRAN77 Syntax Example

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

arg1(1)=TRUE;
call gescape(fildes,GR2D_CONVEX_POLYGONS,arg1,arg2)
```

Pascal Syntax Example

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

var
arg1, arg2 : gescape_arg;

begin
arg1.i[1]:=TRUE;
gescape(fildes,GR2D_CONVEX_POLYGONS,arg1,arg2);
```

GR2D_DEF_MASK

The ⟨*op*⟩ parameter is `GR2D_DEF_MASK`.

Refer to the table, **Supported Device Drivers**, at the front of this chapter for devices that support this gescape.

These devices 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 8x8 squares in the first plane.

C Syntax Example

```
/* 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 Example

```
      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 Example

```
{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 you 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 Example

```
/* 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 Example

```
      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 Example

```
{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`.

Refer to the table, **Supported Device Drivers**, at the front of this chapter for devices that support this gescape.

These devices 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 you 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 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 Example

```
/* 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 Example

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

Pascal Syntax Example

```
{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`.

Refer to the table, **Supported Device Drivers**, at the front of this chapter for devices that support this gescape.

These devices 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 A-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 A-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 A-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 Example

```
/* 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 Example

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

Pascal Syntax Example

```
{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`.

Refer to the table, **Supported Device Drivers**, at the front of this chapter for devices that support this gescape.

These devices 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 monochrome display.

C Syntax Example

```
/* 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 Example

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

Pascal Syntax Example

```
{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 on using `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 Example

```
/* 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 Example

```
integer*4 arg1(64),arg2(64)
arg1.i(1) = 5
```

```
call gescape(fildes,GR2D_PLANE_MASK,arg1,arg2)
```

Pascal Syntax Example

```
{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_PLANE_RULE

The ⟨*op*⟩ parameter is `GR2D_PLANE_RULE`.

This `gescape` specifies independent replacement rules per frame buffer plane for bit/pixel block writes. Source, destination, and pattern values provide the three operands for the replacement rule. The per plane replacement rules are relevant when `R_BIT_MODE` has enabled bit/pixel mode and ⟨*raw*⟩ mode is used in `block_write`. The `gescape` takes two integers. `arg1.i[0]` is the 8 bit replacement rule, see `PATTERN_FILL` for a replacement rule truth table. `arg1.i[1]` is an 8 bit mask indicating which frame buffer plane(s) should use this replacement rule. The default replacement rule is `SOURCE` (rule 0x33) for all frame buffer planes.

The per plane replacement rules are used until a call to `GR2D_MASK_RULE` is made.

A This `gescape` can be called multiple times to define different replacement rules for different frame buffer planes.

The following examples enable the `SOURCE` replacement rule for planes 0,1,2,3 and `DESTINATION` replacement rule for planes 4,5,6,7.

C Syntax Example

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

gescape_arg arg1, arg2;
   ⋮
arg1.i[0]=0x33;
arg1.i[1]=0x0f;
gescape(fildes,GR2D_PLANE_RULE,&arg1,&arg2);
arg1.i[0]=0x55;
arg1.i[1]=0xf0;
gescape(fildes,GR2D_PLANE_RULE,&arg1,&arg2);
```

FORTRAN77 Syntax

```
integer*4 arg1(64),arg2(64)
arg1(1)=51
arg1(2)=15
call gescape(fildes,GR2D_PLANE_RULE,arg1,arg2)
```

```
arg1(1)=85
arg1(2)=240
call gescape(fildes,GR2D_PLANE_RULE,arg1,arg2)
```

Pascal Syntax Example

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

var
   arg1,arg2:gescape_arg;
       :
begin
   arg1.i[1] := 51;
   arg1.i[2] := 15;
   gescape(fildes,GR2D_PLANE_RULE,arg1,arg2);
   arg1.i[1] := 85;
   arg1.i[2] := 240;
   gescape(fildes,GR2D_PLANE_RULE,arg1,arg2);
```

GR2D_REPLICATE

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

Refer to the table, **Supported Device Drivers**, at the front of this chapter for devices that support this gescape.

These devices 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 Example

```
/* 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 Example

```
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 Example

```
{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 your 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 your program has *not* sent an HP-GL output command `O*` 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);

      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);
```

A

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.
```

HPGL2_ADAPTIVE_LINES

The ⟨*op*⟩ parameter is `HPGL2_ADAPTIVE_LINES`.

The `arg1` parameter contains a single flag. If `TRUE`, adaptive line types are enabled; if `False`, adaptive lines are disabled.

The `arg2` parameter is ignored.

Adaptive line types scale the repeat pattern fitting an integer number of patterns between line segment endpoints. Adaptive line types are more suited to drafting standards because the endpoints are seen.

Fixed line types "wrap" the repeat pattern around the sides of a polygon without scaling. Fixed line types may not draw the endpoints because the pattern is in the "move" rather than the "draw" region.

Note Arcs, circles, etc. drawn by Starbase using many tiny line segments will look like solid lines rather than the selected line type when adaptive line types are enabled. Therefore, a thorough understanding of your application and enable/disable adaptive line types is suggested.

C Syntax Example

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

gescape_arg arg1, arg2;

/* Enable adaptive line types */

arg1.i[0] = TRUE;
gescape(fildes,HPGL2_ADAPTIVE_LINES,&arg1,&arg2);
```

FORTRAN77 Syntax Example

```
        integer*4 arg1i(64), arg2i(64)
C
C Enable adaptive line types
C
        arg1i(1) = 1
        call gescape(fildes,HPGL2_ADAPTIVE_LINES,arg1i,arg2i)
```

Pascal Syntax Example

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

begin

    { Enable adaptive line types }

    arg1.i[1] := 1;
    gescape(fildes,HPGL2_ADAPTIVE_LINES,arg1,arg2);
```

HPGL2_CUTTER_CONTROL

The ⟨*op*⟩ parameter is `HPGL2_CUTTER_CONTROL`.

This `gescape` will enable/disable the paper cutter.[1]

The `arg1` parameter contains a single flag. If `TRUE`, the cutter is enabled; if `False`, the cutter is disabled.

The `arg2` parameter is ignored.

C Syntax Example

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

gescape_arg arg1, arg2;

/* Enable cutter */

arg1.i[0] = TRUE;
gescape(fildes,HPGL2_CUTTER_CONTROL,&arg1,&arg2);
```

FORTRAN77 Syntax Example Example

```
    integer*4 arg1i(64), arg2i(64)
C
C Enable cutter
C
    arg1i(1) = 1
    call gescape(fildes,HPGL2_CUTTER_CONTROL,arg1i,arg2i)
```

[1] Some devices may not have a paper cutter. Consult the device's reference manual.

Pascal Syntax Example

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

var
    arg1, arg2 : gescape_arg;

begin

    { Enable cutter }

    arg1.i[1] := 1;
    gescape(fildes,HPGL2_CUTTER_CONTROL,arg1,arg2);
```

HPGL2_FONT_POSTURE

The ⟨*op*⟩ parameter is HPGL2_FONT_POSTURE.

The single integer argument in arg1 indicates the desired font posture. In HP-GL/2 there are two choices:

- 0—Upright (default)
- 1—Italic

The font posture is independent of Starbase.

The arg2 parameter is ignored.

C Syntax Example

```
#define NORMAL_POSTURE 0
#define ITALIC_POSTURE 1

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

gescape_arg arg1, arg2;

/* Select italic posture */

arg1.i[0] = ITALIC_POSTURE;
gescape(fildes,HPGL2_FONT_POSTURE,&arg1,&arg2);
```

FORTRAN77 Syntax Example

```
    integer*4 arg1i(64), arg2i(64)
C
C Select italic posture
C
    arg1i(1) = 1
    call gescape(fildes,HPGL2_FONT_POSTURE,arg1i,arg2i)
```

Pascal Syntax Example

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

begin

    { Select italic posture }

    arg1.i[1] := 1;
    gescape(fildes,HPGL2_FONT_POSTURE,arg1,arg2);
```

HPGL2_FONT_TYPEFACE

The ⟨*op*⟩ parameter is `HPGL2_FONT_TYPEFACE`.

This `gescape` allows a selection from more than 80 font typefaces supported by HP-GL/2; however, this function is dependent on which typefaces are present in the device via soft fonts or cartridge. The font must be present in order to select it.

The `arg1` parameter contains the integer coresponding to the desired typeface.[2]

The `arg2` parameter is ignored.

C Syntax Example

```
#define PRESENTATIONS 11

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

/* Select presentations typeface */

arg1.i[0] = PRESENTATIONS;
gescape(fildes,HPGL2_FONT_TYPEFACE,&arg1,&arg2);
```

FORTRAN77 Syntax Example

```
    integer*4 arg1i(64), arg2i(64)
C
C Select presentation typeface
C
    arg1i(1) = 11
    call gescape(fildes,HPGL2_FONT_TYPEFACE,arg1i,arg2i)
```

Pascal Syntax Example

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

[2] See the table "Typefaces" earlier in this chapter for recognized typefaces for HP-GL/2.

```
var
    arg1, arg2 : gescape_arg;

begin

    { Select presentations typeface }

    arg1.i[1] := 11;
    gescape(fildes,HPGL2_FONT_TYPEFACE,arg1,arg2);
```

HPGL2_FONT_WEIGHT

The ⟨*op*⟩ parameter is `HPGL2_FONT_WEIGHT`.

This `gescape` enables the font stroke weight to be set independent of Starbase.

The `arg1` parameter indicates the single integer argument for the weight number as defined in the HP-GL/2 language. Weight numbers range from −7 (very light) to 0 (normal) to +7 (very bold). Using 9999 when the stick font typeface is selected will cause the current pen width to be used.

The `arg2` parameter is ignored.

C Syntax Example

```
#define MEDIUM_BOLD 3

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

gescape_arg arg1, arg2;

/* Select medium bold weight */

arg1.i[0] = MEDIUM_BOLD;
gescape(fildes,HPGL2_FONT_WEIGHT,&arg1,&arg2);
```

FORTRAN77 Syntax Example

```
    integer*4 arg1i(64), arg2i(64)
C
C Select medium bold weight
C
    arg1i(1) = 3
    call gescape(fildes,HPGL2_FONT_WEIGHT,arg1i,arg2i)
```

Pascal Syntax Example

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

var
    arg1, arg2 : gescape_arg;

begin

    { Select medium bold weight }

    arg1.i[1] := 3;
    gescape(fildes,HPGL2_FONT_WEIGHT,arg1,arg2);
```

HPGL2_LOGICAL_PEN_WIDTH

The ⟨*op*⟩ parameter is `HPGL2_LOGICAL_PEN_WIDTH`.

Note Logical pen width provides a wideline capability separate from Starbase widelines.

The `arg1` parameter is the pen number or color map entry.[3]

The `arg2` parameter indicates the width in millimeters.

Each pen can be set to a different logical width. If the pen number parameter in `arg1` is `-1`, all the pens are set to that width. The device will determine the physical pen width and make the appropriate number of strokes to emulate the logical pen width.

C Syntax Example

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

gescape_arg arg1, arg2;

/* Set pen #3 to stroke out 6.0 mm lines */

arg1.i[0] = 6.0;
arg2.i[0] = 3;
gescape(fildes,HPGL2_LOGICAL_PEN_WIDTH,&arg1,&arg2);
```

[3] Whether `arg1` represents a physical pen number or a color map entry depends on the device. Electrostatic plotters have no physical pens.

FORTRAN77 Syntax Example

```
        real arg1f(64)
        integer*4 arg2i(64)
    C
    C Set pen #3 to stroke out 6.0 mm lines
    C
        arg1f(1) = 6.0
        arg2i(1) = 3
        call gescape(fildes,HPGL2_LOGICAL_PEN_WIDTH,arg1i,arg2i)
```

Pascal Syntax Example

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

    var
        arg1, arg2 : gescape_arg;

    begin

        { Set pen #3 to stroke out 6.0 mm lines }

        arg1.f[1] := 6.0;
        arg1.i[1] := 3;
        gescape(fildes,HPGL2_LOGICAL_PEN_WIDTH,arg1,arg2);
```

HPGL2_REPLOT

The ⟨*op*⟩ parameter is `HPGL2_REPLOT`.

This `gescape` allows you to replot the command buffer, eliminating the need to re-transmit data for each copy.

The `arg1` parameter contains the number of replots (copies) desired.

The `arg2` parameter is ignored.

C Syntax Example

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

gescape_arg arg1, arg2;

/* Make 2 copies of the plot */

arg1.i[0] = 2;
gescape(fildes,HPGL2_REPLOT,&arg1,&arg2);
```

FORTRAN77 Syntax Example

```
    integer*4 arg1i(64), arg2i(64)
C
C Make 2 copies of the plot
C
    arg1i(1) = 2
    call gescape(fildes,HPGL2_REPLOT,arg1i,arg2i)
```

Pascal Syntax Example

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

var
    arg1, arg2 : gescape_arg;

begin

    { Make 2 copies of the plot }

    arg1.i[1] := 2;
    gescape(fildes,HPGL2_REPLOT,arg1,arg2);
```

HPGL2_SET_CMAP_SIZE

The ⟨*op*⟩ parameter is HPGL2_SET_CMAP_SIZE.

This gescape allows you to resize the default color map. It can be used as many times as needed for electrostatic plotters; however, for pen plotters, use this gescape once at the beginning of the application to set the color map to the number of physical pens in the carousel.

The arg1 parameter contains the color map size.

The arg2 parameter is ignored.

Resizing the color map will de-allocate the current color map. Therefore, all changes to the color map entries made by the Starbase call define_color_table will be lost. The color map entries are re-initialized to their default values.

C Syntax Example

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

gescape_arg arg1, arg2;

/* My pen plotter only has 8 pens + 1 (no pen) = 9 */

arg1.i[0] = 9;
gescape(fildes,HPGL2_SET_CMAP_SIZE,&arg1,&arg2);
```

FORTRAN77 Syntax Example

```
    integer*4 arg1i(64), arg2i(64)
C
C My pen plotter only has 8 pens + 1 (no pen) = 9
C
    arg1i(1) = 9
    call gescape(fildes,HPGL2_SET_CMAP_SIZE,arg1i,arg2i)
```

Pascal Syntax Example

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

var
    arg1, arg2 : gescape_arg;

begin

    { My pen only has 8 pens +1 (no pen) = 9 }

    arg1.i[1] := 9;
    gescape(fildes,HPGL2_SET_CMAP_SIZE,arg1,arg2);
```

HPGL2_SET_MEDIA_TYPE

The ⟨*op*⟩ parameter is `HPGL2_SET_MEDIA_TYPE`.

This `gescape` allows you to choose from various types of media. The plotter will optimize plotting speed and force based on the media selected.

The `arg1` parameter can contains the following integers:

0—paper
1—transparency
2—velum
3—polyester film
4—translucent paper
5—special paper

The `arg2` parameter is ignored.

C Syntax Example

```
#define PAPER        0
#define TRANSPARENCY 1

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

gescape_arg arg1, arg2;

/* Set to plot on a transparency */

arg1.i[0] = TRANSPARENCY;
gescape(fildes,HPGL2_SET_MEDIA_TYPE,&arg1,&arg2);
```

FORTRAN77 Syntax Example

```
          integer*4 arg1i(64), arg2i(64)
     C
     C Set to plot on a transparency
     C
          arg1i(1) = 1
          call gescape(fildes,HPGL2_SET_MEDIA_TYPE,arg1i,arg2i)
```

Pascal Syntax Example

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

     var
         arg1, arg2 : gescape_arg;

     begin

         { Set to plot a transparency }

         arg1.i[1] := 1;
         gescape(fildes,HPGL2_SET_MEDIA_TYPE,arg1,arg2);
```

HPGL2_SET_QUALITY

The ⟨*op*⟩ parameter is `HPGL2_SET_QUALITY`.

This `gescape` uses a single integer parameter in `arg1` between `0` and `100`. The integer indicates the desired quality level of the output. The primary effect is on pen velocity.

The `arg2` parameter is ignored.

Note This `gescape` should only be invoked once per plot—after a rasterizing command (`PG` or `RP`) of the previous plot and before the first command that causes marks on the media for the current plot. Use of this `gescape` during the plot body will result in an error.

C Syntax Example

```
#define DRAFT_COPY        0
#define MEETING_COPY      50
#define FINAL_COPY        100

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

gescape_arg arg1, arg2;

/* Need a pretty good copy to present */

arg1.i[0] = MEETING_COPY;
gescape(fildes,HPGL2_SET_QUALITY,&arg1,&arg2);
```

FORTRAN77 Syntax Example

```
      integer*4 arg1i(64), arg2i(64)
C
C Need a pretty good copy to present
C

      arg1i(1) = 50
      call gescape(fildes,HPGL2_SET_QUALITY,arg1i,arg2i)
```

Pascal Syntax Example

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

var
    arg1, arg2 : gescape_arg;

begin

    { Need a pretty good copy to present }

    arg1.i[1] := 50;
    gescape(fildes,HPGL2_SET_QUALITY,arg1,arg2);
```

HPTERM_640x400

The ⟨*op*⟩ parameter is `HPTERM_640x400`.

The graphics display resolution for the HP 2393 and the HP 2397 is normally determined at `gopen` time with a terminal inquiry. However, such an inquiry is impossible when the output is spooled. In this case, a resolution of 512×390 is assumed. This `gescape` is provided to change the transformation matrix to use the 640×400 resolution possible with these two terminals.

The `gescape` call should immediately follow the call to `gopen` and should be followed by an appropriate call to `set_p1_p2` to reset the transformation matrix.

Both `arg1` and `arg2` are ignored.

C Syntax Example

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

int fildes;

fildes = gopen("/dev/tty", OUTDEV, "hp2393", INIT | SPOOLED);
gescape(fildes, HPTERM_640x400, 0, 0);
set_p1_p2(fildes, FRACTIONAL, 0.0, 0.0, 0.0, 1.0, 1.0, 1.0);
```

FORTRAN77 Syntax Example

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

fildes = gopen("/dev/tty"//char(0), OUTDEV, "hp2393"//char(0),
        INIT | SPOOLED)
call gescape(fildes,HPTERM_640x400,arg1,arg2)
set_p1_p2(fildes,FRACTIONAL,0.0,0.0,0.0,1.0,1.0.1.0)
```

Pascal Syntax Example

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

var
  fildes:integer;
  arg1,arg2:gescape_arg

begin
  fildes := gopen("/dev/tty", OUTDEV, "hp2393", INIT | SPOOLED);
  gescape(fildes, HPTERM_640x400, 0, 0);
  set_p1_p2(fildes, FRACTIONAL, 0.0, 0.0, 0.0, 1.0, 1.0, 1.0);
```

HPTERM_PRINT_ESC

The ⟨*op*⟩ parameter is HPTERM_PRINT_ESC or HP26_PRINT_ESC.

NULL-terminated strings (those which end with a char(0)) containing terminal escape sequences can be sent to the terminal using this gescape.

The ar g1 parameter is the string you wish to send.

The arg2 parameter is ignored.

The following examples show one way to clear the alpha display.

C Syntax Example

```
 #include <starbase.c.h>
 main()
 {
    int               fildes, status;
    gescape_arg       arg2, sequence;
/* gescape_arg from starbase.c.h */
    strcpy(sequence.c, "\033h\033J");
    fildes = gopen("/dev/tty", OUTDEV, "hpterm", INIT);
    gescape(fildes, HPTERM_PRINT_ESC, &sequence, &arg2);
    status = gclose(fildes);
 }
```

FORTRAN77 Syntax Example

```
 include '/usr/include/starbase.f1.h'
 program gesc
 integer*4 fildes, status
 include '/usr/include/starbase.f2.h'
 fildes = gopen('/dev/tty'//char(0),
     OUTDEV, 'hpterm'//char(0), INIT)
 call gescape(fildes, HPTERM_PRINT_ESC,
+ char(27)//'h'//char(27)//'J'//char(0), ' ')
 status = gclose(fildes)
 end
```

Pascal Syntax Example

```
program gesc;
$include '/usr/include/starbase.p1.h'$
var
  fildes, status:    integer;
  arg2, sequence:    gescape_arg;
$include '/usr/include/starbase.p2.h'$

begin
  fildes := gopen('/dev/tty', OUTDEV, 'hpterm', INIT);
  sequence.c[1] := chr(27);
  sequence.c[2] := 'h';
  sequence.c[3] := chr(27);
  sequence.c[4] := 'J';
  sequence.c[5] := chr(0);
  gescape(fildes, HPTERM_PRINT_ESC, sequence, arg2);
  status := gclose(fildes);
end.
```

IGNORE_PROXIMITY

The ⟨*op*⟩ parameter is IGNORE_PROXIMITY.

This gescape causes the device not to generate a choice input and a locator input when the device's stylus is close enough to the device to register input activities. This is the default state.

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_PROXIMITY, &arg1, &arg2);
```

A

FORTRAN77 Syntax Example

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

Pascal Syntax Example

```
{gescape_arg is defined in starbase.p1.h}
var
   arg1, agr2: gescape_arg;
begin
   gescape (fildes, IGNORE_PROXIMITY, arg1, arg2);
```

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);
```

ILLUMINATION_ENABLE

This gescape allows you to specify the amount of data that is present per vertex when `ILLUMINATION` is specified as extra data for the “with data” primitives. The default number of words to use is two (2). It is also used to specify if the illumination data is to be used.

`arg1.i[0]` is set to the number of words per vertex to use for illumination data. `arg1.i[1]` is set to `TRUE` or `FALSE` specifying whether the illumination data should be used.

C Syntax Example

```
gescape_arg arg1, arg2;
        ⋮
arg1.i[0]=2;
arg1.i[1]=TRUE;
gescape(fildes,ILLUMINATION_ENABLE,&arg1,&arg2);
}
```

FORTRAN77 Syntax Example

```
integer*4 arg1(64),arg2(64)
        ⋮
arg1.i[0]=2
arg1.i[1]=TRUE
call gescape(fildes,ILLUMINATION_ENABLE,arg1,arg2)
```

Pascal Syntax Example

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

var
   arg1,arg2:gescape_arg;
begin
        :
   arg1.i[0] := 2;
   arg1.i[1] := TRUE;
   gescape(fildes,ILLUMINATION_ENABLE,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 you 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 calls to the `SET_BANK_CMAP` `gescape` and `define_color_table`. 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 you 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 Example

```
/* 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 Example

```
      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 Example

```
      {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
```

INQ_12BIT_INDEXING

The ⟨*op*⟩ parameter is `INQ_12BIT_INDEXING`.

This `gescape` takes no input parameters and returns in `arg2.i[0]` a 1 if the current display mode is 12 bit indexing and a 0 otherwise.

The following examples determine whether or not 12 bit indexing is the current display mode.

C Syntax Example

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

gescape_arg arg1, arg2;
   ⋮
gescape(fildes,INQ_12BIT_INDEXING,&arg1,&arg2);
if (arg2.i[0] == TRUE)
{
        printf("12 bit indexing is on.");
}
else
{
        printf("12 bit indexing is off.");
}
```

FORTRAN77 Syntax Example

```
integer*4 arg1(64),arg2(64)
call gescape(fildes,INQ_12BIT_INDEXING,arg1,arg2)
if (arg2[1] .eq. 1) then
        write(6,*)'12 bit indexing is on.'
else
        write(6,*)'12 bit indexing is off.'
endif
```

Pascal Syntax Example

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

var
   arg1,arg2:gescape_arg;
       ⋮
begin
   gescape(fildes,INQ_12BIT_INDEXING,arg1,arg2);
if (arg2.i[1] = 1) then
        write('12 bit indexing is on.')
else
        write('12 bit indexing is off.');
```

LINEAR_POLYPOINT

The LINEAR_POLYPOINT gescape provides the means to specify points to be rendered along a line specified in modeling coordinates. For each point specified by a data value, one pixel will be rendered.

The data may be provided in any orientation relative to the device coordinate (DC) space. This entry point sorts the data so as to always render the points from back to front in the DC Z axis. The data specified is composited with the current contents of the frame buffer.

This primitive assumes as 24-bit CMAP_FULL shade-mode (or 12/12 CMAP_FULL double-buffer).

The compositing function is of the form:

A

```
        new value = (alpha * source) + ( (1 - alpha) * destination)
```

where source is the supplied value and destination is the current frame buffer contents. Compositing is performed separately for the red, green, and blue banks.

This gescape does not support perspective transformations.

The arg2 parameter is ignored.

Input Parameters:

Start of the line

arg1.f[0]	X value in Modeling coordinates
arg1.f[1]	Y value in Modeling coordinates
arg1.f[2]	Z value in Modeling coordinates

End of the line

arg1.f[3]	X value in Modeling coordinates
arg1.f[4]	Y value in Modeling coordinates
arg1.f[5]	Z value in Modeling coordinates
arg1.i[6]	Number of points to render
arg1.i[7]	Number of voxels (data values) per point to skip

`arg1.i[8]`	Vertex format flags
`arg1.i[9]`	pointer to the data values (long words, packed bytes, or packed shorts)
`arg1.i[10]`	pointer to the data mapping table scalar(byte) -> alpha,r,g,b (this parameter applies only to the INDIRECT vertex formats)

The allowed values for the vertex format flag are:

`NULL`	Standard format is a pack word per voxel. Each word contains an byte of: (alpha, red, green, blue) with each byte scaled in the range of 0-255.
`INDIRECT_DATA`	Indicates that the vertex format is a packed byte array of monotonic intensities which map into a 256 entry (alpha, red, green, blue) lookup table supplied via a pointer in arg1.i[20].
`INDIRECT_DATA16`	Indicates that the vertex format is a packed short (16 bit) array of intensity which maps into a 65,536 entry (alpha, red, green, blue) lookup table supplied via a pointer in arg1.i[20].
`PRE_SCALED_ALPHA`	Indicates that the vertex format is per the CRX-24/24Z's fast path (1-alpha, alpha*r, alpha*g, alpha*b). The CRX-24 will operate most efficiently (quickly) if the direct data or indirect table is pre-scaled as described here. This pre-scaling will avoid having the gescape scale the data per voxel (for the DIRECT case) or per gescape (for the INDIRECT case). Note that the CRX-48Z can directly handle alpha, red, green, and blue. For the CRX-48Z this option is slower and should not be used, although it will still work.

LS_OVERFLOW_CONTROL

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

Refer to the table, **Supported Device Drivers**, at the front of this chapter for devices that support this gescape.

Four options are provided to you 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`

A

`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 you to quickly determine where the light source equations are overflowing. You select 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 you 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.

Similar functionality to the `HYBRID` option is available on other devices with `surface_coefficients`.

The following examples select a scaled model.

C Syntax Example

```
/*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 Example

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

Pascal Syntax Example

```
{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);
```

OLD_SAMPLE_ON_DIFF_SCREEN

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

This gescape inquires the locator and choice sampling of the X11 pointer device.

The X server's locator position can be sampled anytime, and is returned relative to the window. By default, when the X pointer is on a different screen than the window, the valid parameter of the sample_locator procedure is returned as FALSE.

When the gescape `OLD_SAMPLE_ON_DIFF_SCREEN` is used, the valid parameter is returned as TRUE when the X pointer is on a different screen. In this case, the pointer position returned by sample_locator is either the last value of the X locator on that screen or the value (0,0) if the pointer has never been on that screen.

To restore the default (valid set to FALSE) behavior, use the `BAD_SAMPLE_ON_DIFF_SCREEN` gescape.

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.

A

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 Example

```
/*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 Example

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

Pascal Syntax Example

```
{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

The ⟨*op*⟩ parameter is `PAN`.

The `PAN gescape` is only supported in the image planes.

Pan is a function that places the video display in a 1024 x 1024 output mode and allows the upper left corner of the video scan to be either at address 0, 0 in the frame buffer or address 1024, 0.

That is, the video output of the display can be made to show the contents of either the left or right half of the 2048x1024 frame buffer.

This `gescape` can be used with the HP 98735 and HP 98736 device drivers to control the display hardware.

To activate the 1024 x 1024 display mode the gescape should be called with: `arg1.i[0]=1`.

To display the left half of the frame buffer (address 0,0 in the upper left corner) the gescape should be called with: `arg1.i[1]=0`. To display the right half (address 1024,0 in the upper left corner) the gescape should be called with: `arg1.i[1]=1`.

Setting `arg1.i[0]=0` will deactivate the 1024 x 1024 display mode and return the upper left corner to it normal position.

The following examples activates the pan function to the left side of the frame buffer with the first `gescape` call and then shifts it to the right side of the frame buffer with the second `gescape` call.

C Syntax Example

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

gescape_arg arg1, arg2;
    ⋮
arg1.i[0]=1;
arg1.i[1]=0;
gescape(fildes,PAN,&arg1,&arg2); /* left */
arg1.i[1]=1;
gescape(fildes,PAN,&arg1,&arg2); /* right */
```

FORTRAN77 Syntax Example

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

Pascal Syntax Example

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

var
   arg1,arg2:gescape_arg;
       ⋮
begin
   arg1.i[1] := 1;
   arg1.i[2] := 0;
   gescape(fildes,PAN,arg1,arg2); { left }
   arg1.i[2] := 1;
   gescape(fildes,PAN,arg1,arg2); { right }
```

PAN_AND_ZOOM

The $\langle op \rangle$ 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 Example

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

  gescape_arg arg1, arg2;
     :
  arg1.i[0]=1;  /* Request PANX,PANY represent pixel to become center*/
                /* of the screen */
  arg1.i[1]=128;
  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 Example

```
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 Example

```
{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 A-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 A-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 `draw-ing_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.

Note For the HP 98736 device, the maximum value for `dx` and `dy` is 32. For patterns larger than 32×32, use texture maps.

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 Example

```
/*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 Example

```
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 Example

```
{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);
```

PLUG_ACCELERATED_PIPELINE

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

The PLUG_ACCELERATED_PIPELINE gescape controls the rendering of the graphics accelerators into the frame buffer. It is necessary to stop the accelerators from rendering when accessing the frame buffer directly as memory locations mapped into a program's address space (see the R_GET_FRAME_BUFFER gescape). It is necessary to restart the accelerators before making Starbase calls via the HP 98736 and HP CRX-48Z device drivers.

This `gescape` is supported by the HP 98736 and HP CRX-48Z device drivers.

Setting `arg1.i[0]=1` will stop the accelerators while setting `arg1.i[0]=0` will restart them.

The following example illustrates the intermixing of Starbase calls and direct frame buffer access.

C Syntax Example

```
#include ⟨starbase.c.h⟩

main(argc,argv)
int argc;
char **argv;
{
        register int fildes,i;
        register unsigned int *frame;
        gescape_arg arg1,arg2;

        /* Open the 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 int *) arg2.i[1];

        /* Lock the device and plug the accelerators before frame buffer access */
        gescape(fildes,R_LOCK_DEVICE,&arg1,&arg2);
        arg1.i[0]=1;
        gescape(fildes,PLUG_ACCELERATED_PIPELINE,&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] = 3;

        /* Unplug the accelerators and unlock the device */
        arg1.i[0]=0;
        gescape(fildes,PLUG_ACCELERATED_PIPELINE,&arg1,&arg2);
        gescape(fildes,R_UNLOCK_DEVICE,&arg1,&arg2);

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

FORTRAN77 Syntax Example

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       Plug the accelerators
        arg1(0)=1
        call gescape(fildes,PLUG_ACCELERATED_PIPELINE,arg1,arg2)

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

C       Unplug the accelerators
        arg1(0)=0
        call gescape(fildes,PLUG_ACCELERATED_PIPELINE,arg1,arg2)

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 Example

```
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);

  { Plug the accelerators}
  arg1.i[0]:= 1;
  gescape(fildes,PLUG_ACCELERATED_PIPELINE,arg1,arg2);

  { Draw line from (x=99,y=5) to (x=99,y=299) }
  for i := 5 to 299 do
    frame^[99 + i*2048] := chr(3);
  { Unplug the accelerators}
  arg1.i[0]:= 0;
  gescape(fildes,PLUG_ACCELERATED_PIPELINE,arg1,arg2);

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

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

POLYGON_TRANSPARENCY

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

This `gescape` allows you to define separate "screen door" transparency patterns for frontfacing and backfacing polygons. You may define patterns that disable writes to any pixels within a 4×4 cell. This cell is duplicated over the entire screen.

You pass in a bit mask where a 1 means the corresponding pixel is write enabled and a 0 means write disabled. Table 1-5 shows the 2 byte bit pattern that is passed in by you, and table 1-6 shows how that pattern is turned into a 4×4 dither pattern.

The `arg1[0]` parameter contains the mask to be used for front facing polygons.

The `arg1[1]` parameter contains the mask to be used for back facing polygons.

Table A-8.

15	...	2	1	0

Table A-9.

0	1	2	3
4	5	6	7
8	9	10	11
12	13	14	15

The following examples would produce a green square with a 50% transparent red rectangle in front. Backfacing polygons remain opaque. Remember to set both of the transparency patterns back to opaque when done.

C Syntax Example

```
/* 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;
arg1.i[1] = 0xFFFF;
gescape(fildes,POLYGON_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;
arg1.i[1] = 0xFFFF;
gescape(fildes,POLYGON_TRANSPARENCY,&arg1,&arg2);
```

FORTRAN77 Syntax Example

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

fill_color(fildes,0.0,1.0,0.0);
rectangle(fildes,0.25,0.25,0.75,0.75);
arg1(1)=Z'AAAA'
arg1(2)=Z'FFFF'
call gescape(fildes,POLYGON_TRANSPARENCY,arg1,arg2)
fill_color(fildes,1.0,0.0,0.0);
rectangle(fildes,0.0,0.25,1.0,0.75);
arg1(1)=Z'FFFF'
arg1(2)=Z'FFFF'
call gescape(fildes,POLYGON_TRANSPARENCY,arg1,arg2)
```

Pascal Syntax Example

```
{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');
   arg1.i[2] := hex('FFFF');
   gescape(fildes,POLYGON_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');
   arg1.i[2] := hex('FFFF');
   gescape(fildes,POLYGON_TRANSPARENCY,arg1,arg2);
```

PROMPT_OFF

The ⟨*op*⟩ parameter is `PROMPT_OFF`.

This `gescape` manually deactivates the prompt indicator on the specified device (if the device has one).

The `arg1` and `arg2` parameters are ignored.

C Syntax Example Example

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

FORTRAN77 Syntax Example

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

Pascale Syntax Example

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

PROMPT_ON

The ⟨*op*⟩ parameter is `PROMPT_ON`.

This `gescape` manually activates the prompt indicator on the specified device (if the device has one).

The `arg1` and `arg2` parameters are ignored.

C Syntax Example Example

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

FORTRAN77 Syntax Example

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

Pascale Syntax Example

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

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 Example

```
/* 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 Example

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

Pascal Syntax Example

```
{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 Example

```
/* 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 Example

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

Pascal Syntax Example

```
{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 98705, HP 98730, HP 98731, HP 98735, HP 98736 and HP CRX-48Z devices, use `R_ECHO_MASK` for slightly better performance.

With the HP 98704, HP 98730, and the HP 98735 device drivers, 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 Example

```
/* 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 Example

```
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 Example

```
{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 you 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 Example

```
/* 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 Example

```
      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 Example

```
{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 the 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 your responsibility to properly align your 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 using 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 Example

```
#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 aligned 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 Example

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 Example

```
$standard_level 'hp_modcal'$
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`.

Refer to the table, **Supported Device Drivers**, at the front of this chapter for devices that support this gescape.

These devices provide 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. The driver makes only one attempt to use the hardware cursor. 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 you know 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 Example

```
/* 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 Example

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

Pascal Syntax Example

```
{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`

Refer to the table, **Supported Device Drivers**, at the front of this chapter for devices that support this gescape.

These devices 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. Software cursors are cursors that may be 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 lowers performance.

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

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

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. Therefore, 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 all devices. 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 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

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

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 you 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 HP 98704, HP 98730 and HP 98735 or HP CRX-48Z devices, this `gescape` should be used for better performance.

With the `hp98730` and the `hp98704` device drivers, echo masks cannot be used in a graphics window if the echo currently being used is not the hardware cursor and 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 10x8 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 Example

```
/* 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 Example

```
integer*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 Example

```
 {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. X11 also uses offscreen for its fonts and sprite, as well as pixmaps and backing store (retained rasters). Notice: X11 leaves you 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 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 and X11.

C Syntax Example

```
/* 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 Example

```
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 Example

```
{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.

Note Be careful when accessing the frame buffer directly via this gescape to alter the contents of an X11 window. This gescape allows access to the *entire* frame buffer. Therefore, it is possible to alter the contents of windows that obscure (i.e. are on top of) the window of interest.

Also be aware that the window position, size, or stack location (top, bottom, etc.) may change between the time you determine these characteristics and the time that you lock the frame buffer via the `R_LOCK_DEVICE` gescape.

Device Dependency

These examples are written for the Series 300/400 and 700 computers. On the Series 800 computers, HP 98735 and HP 98736 devices, the frame buffer arrays must be converted to arrays of integers since the I/O Bus is 32 bits wide. In the C syntax example, the line `register unsigned char *frame` would become `register unsigned int *frame`. The HP 98736 and HP CRX-48Z must halt accelerators via the `PLUG_ACCELERATED_PIPELINE gescape` before the frame buffer may be accessed.

C Syntax Example

```
#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 Example

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 Example

```
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_LINE_TYPE

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

Refer to the table, **Supported Device Drivers**, at the front of this chapter for devices that support this gescape.

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 Example

```
/* 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 Example

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

A

Pascal Syntax Example

```
{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 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.
- If the `hp98735` and `hp98736` drivers are opened concurrently, the `hp98735` driver will plug the accelerated pipeline when this `gescape` is called. As a result, no `hp98736` commands will be processed.

C Syntax Example

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

gescape_arg arg1, arg2;

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

FORTRAN77 Syntax Example

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

Pascal Syntax Example

```
{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 X11. Starbase and X11 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 X11 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 98705, HP 98730, HP 98735 devices, 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 Example

```
/* 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 Example

```
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 Example

```
{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 Example

```
/* 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 Example

```
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 Example

```
{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`.

Refer to the table, **Supported Device Drivers**, at the front of this chapter for devices that support this gescape.

These devices are configured with 3 or 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 98704, HP 98705, HP 98730, HP 98731, HP 98735, and HP 98736

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 Example

```
/* 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 Example

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

Pascal Syntax Example

```
{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 98705, HP 98730, HP 98735

These devices 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 `hp98705`, `hp98731`, and the `hp98736` device drivers always overlay 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 *X11* chapter for more information.

This `gescape` is a `no-op` if the hardware cursor is being used.

C Syntax Example

```
/* 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 Example

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

Pascal Syntax Example

```
{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`.

Refer to the table, **Supported Device Drivers**, at the front of this chapter for devices that support this gescape.

HP 98720 and HP 98721

These devices 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 98705, HP 98730, HP 98731, HP 98735, and HP 98736

These devices come equipped with three or 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 respective driver sections for details.) This `gescape` can be used to set a color map entry to transparent (that is, the color for a pixel is 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 Example

```
/* 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 Example

```
integer arg1(64),arg2(64)
arg1(1)=0
call gescape(fildes,R_TRANSPARENCY_INDEX,arg1,arg2)
```

Pascal Syntax Example

```
{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.

Note The `hp98735` driver will unplug the accelerated pipeline as a result of this `gescape`.

C Syntax Example

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

gescape_arg arg1, arg2;

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

FORTRAN77 Syntax Example

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

Pascal Syntax Example

```
{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 Graphics Techniques* for information on using this `gescape` in an X11 window.

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

C Syntax Example

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

gescape_arg arg1, arg2;

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

FORTRAN77 Syntax Example

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

Pascal Syntax Example

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

var
   arg1,arg2 : gescape_arg;

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

REPORT_PROXIMITY

The ⟨*op*⟩ parameter is REPORT_PROXIMITY.

This gescape causes the device to generate a choice input (with value 8) and a locator input (the locators current position) when the device's stylus comes close enough to the device to register input activities. If TRIGGER_ON_RELEASE is set, the device will also trigger a choice input and a locator input when the device's stylus goes too far away from the device to register inputs. REPORT_PROXIMITY is only supported on HIL devices that have the ability to detect proximity (touch bezels, and some tablets). The default value is for proximity detection to be ignored.

The arg1 and arg2 parameters are ignored.

C Syntax Example Example

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

FORTRAN77 Syntax Example

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

Pascale Syntax Example

```
{gescape_arg is defined in starbase.p1.h}
var
   arg1, agr2: gescape_arg;
begin
   gescape (fildes, REPORT_PROXIMITY, arg1, arg2);
```

SBVESC_BEGIN_ARC and SBVESC_END_ARC

The ⟨*op*⟩ parameters are `SBVESC_BEGIN_ARC` or `SBVESC_END_ARC`.

These `gescape` functions inform the driver when to begin and end archiving Starbase calls.

The function pair provides selective control of Starbase geometry going into any object definition in the archive file. It can be used to "comment out" sections of display code which should not go to the archive. An example of this might be for code which generates more than one view of an object. Additional passes would fill the archive with redundant database information. By carefully placing a pair of gescapes around the other passes to ignore those calls, only one copy of the relevant information is stored in the archive.

By default, the driver is opened with an implied `SBVESC_BEGIN_ARC` call, that is, all relevant Starbase calls are to be archived. To disable archiving for a period of time, a `SBVESC_END_ARC` `gescape` should be issued, with a matching `SBVESC_BEGIN_ARC` to re-enable archiving when it is desired.

C Syntax Example

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

gescape_arg arg1, arg2;
⋮
/* These Starbase calls define "door" object */

gescape(fildes, SBVESC_OBJ_NAME, "door", &arg2);
⋮
/* Ignore all Starbase calls here */

gescape(fildes, SBVESC_END_ARC, &arg1, &arg2);
⋮
/* Continue capturing Starbase calls in "door" object */

gescape(fildes, SBVESC_BEGIN_ARC, &arg1, &arg2);
⋮
```

SBVESC_COMMENT

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

This `gescape` informs the device driver to embed the given user string inside the archive file. Since the archive format is accessible by a standard text editor, embedding useful comments may assist in terms of debugging or noting useful information.

The comment information will be ignored by the translator when it is encountered in the archive file.

C Syntax Example

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

gescape_arg arg2;

⋮

gescape(fildes, SBVESC_COMMENT,
        "Doorknob created Mon Jan 1 00:00:00 MST 2001", &arg2);

⋮
```

SBVESC_LF_COORD

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

This `gescape` informs the device driver that data passed to the driver is in a *left-handed* coordinate system.

This `gescape` is optional. Normally, Starbase expects data passed to it to be in a *left-handed* coordinate system and is thus the default expectation of this driver. However, it is necessary to use this call if *right-handed* coordinate data is intermixed with *left-handed* coordinate data into the data stream. (Refer to `SBVESC_RT_COORD`.) A call of this type to the driver will reset it into a *left-handed* output mode.

By default, the driver starts out with an implied `SBVESC_LF_COORD` (*left-handed*) being issued.

C Syntax Example

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

gescape_arg arg1, arg2;

⋮

gescape(fildes, SBVESC_OBJ_NAME, "doorknob", &arg2);

/* These Starbase calls define "doorknob" object
                  in a left-handed coordinate system. */

gescape(fildes, SBVESC_LF_COORD, &arg1, &arg2);

⋮
```

SBVESC_OBJ_NAME

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

This `gescape` generates a new object definition within the archive under the name given by the `gescape` arguments.

Every time this `gescape` is made, any previous object definitions are completed before beginning the new definition. The name string passed to the `gescape` will be used by the external data translator to create a Personal Visualizer^{TM} object data file with this name.

This `gescape` is optional. If no `gescape` is made, then the external data translator considers all the data in the archive file as one complete object. It will translate the archive's contents into a Personal Visualizer^{TM} compatible file with the same root name as the archive file. Use of this `gescape` separates data going into the archive into unique objects.

Object names can be used multiple times and at different locations in the archive. When processed by the external translator, all information labeled by a common name will be output as a single object. Using this mechanism, model data can be separated by color or group attributes for later manipulation in the Personal Visualizer^{TM}.

The name string passed to the `gescape` must use HP-UX legal filename characters or an error will result when the file is processed by the external translator. Additionally, object names ultimately used by the Personal Visualizer^{TM} are derived from the first 5 letters of the name given in this argument. If these are not unique, then name conflicts may arise when importing this information into the Personal Visualizer^{TM}.

C Syntax Example

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

gescape_arg arg2;

⋮

/* These Starbase calls define object "red" */

gescape(fildes, SBVESC_OBJ_NAME, "red", &arg2);

⋮

/* These Starbase calls define object "blue" */

gescape(fildes, SBVESC_OBJ_NAME, "blue", &arg2);

⋮

/* These Starbase calls add more data to the final object "red" */

gescape(fildes, SBVESC_OBJ_NAME, "red", &arg2);

⋮
```

SBVESC_RT_COORD

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

This `gescape` informs the device driver that data passed to the driver is described in a *right-handed* coordinate system.

This `gescape` is not optional when working with *right-handed* coordinate system data. By default, Starbase expects data passed to it to be in a *left-handed* coordinate system. However, if *right-handed* coordinate data is passed to the driver, additional information must be made available to perform the correct data encoding for the archive.

`SBVESC_RT_COORD` expects the first argument to contain either a 1.0 or -1.0, depending upon transformation matrices normally specified to the `view_matrix()` call. If you normally concatenates an identity matrix with -1.0 in the Z position to flip from *right-handed* space to *left-handed* space (as suggested in the *Starbase Graphics Techniques* tutorial), then this value is passed to the SBV driver as a -1.0. The functionality of `view_matrix()` is disabled by the `hpsbv` driver (since view dependent information is ignored). It is therefore necessary to incorporate this -Z operation elsewhere in the driver if data is to remain correct when converting from the *right-handed* coordinate system.

To toggle the driver back into *left-handed* coordinate space, use the `SBVESC_LF_COORD` `gescape`.

C Syntax Example

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

gescape_arg arg1, arg2;

⋮

/* These Starbase calls define "doorknob" object       */
/* in a left-handed coordinate system by default.  */

gescape(fildes, SBVESC_OBJ_NAME, "doorknob", &arg2);

⋮

/* User normally concatenates -Z matrix to view matrix */

arg1.f[0] = -1.0;

⋮

/* These Starbase calls add data to object
   "doorknob" object which originates from a
    right-handed coordinate system. */

gescape(fildes, SBVESC_RT_COORD, &arg1, &arg2);

⋮

/* These Starbase calls define "doorknob" object
   back in a left-handed coordinate system. */

gescape(fildes, SBVESC_LF_COORD, &arg1, &arg2);

⋮
```

SET_ACCELERATION

The $\langle op \rangle$ parameter is `SET_ACCELERATION`.

This `gescape` causes motion to be multiplied by the acceleration multiplier when the movement per sample (in device coordinates) exceeds the threshold value. The sample rate for HIL is 60 hertz.

`arg1.i[0]`	= *The acceleration multiplier.*
`arg1.i[1]`	= *The threshold value.*

The `arg2` parameter is ignored.

C Syntax Example Example

```
/* gescape_arg is type defined in starbase.c.h */
gescape_arg arg1, arg2;
arg1.i[0]=2;  /* Set acceleration multiplier to 2. */
arg1.i[1]=4;  /* Accelerate when the movement exceeds 4
                 device coordinates per sample. */
gescape(fildes,SET ACCELERATION,&arg1,&arg2);
```

FORTRAN77 Syntax Example

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

Pascale Syntax Example

```
{gescape_arg is defined in starbase.p1.h}
var
   arg1, arg2 : gescape_arg;
begin
   arg1.i[1]:=2;
   arg1.i[2]:=4;
   gescape(fildes,SET_ACCELERATION,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 Example

```
/* 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 Example

```
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 Example

```
{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
```

SET_BUFFER_SIZE

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

This `gescape` allows the application to dynamically change the device driver's internal buffer size. Buffer size can affect application performance. Too large a buffer can affect interactivity. Too small a buffer can add additional overhead to rendering - especially when a Starbase echo (cursor) is active. The ideal buffer size ultimately depends on the application.

This `gescape` will cause the commands currently in the driver's buffer to be executed (flushed).

This `gescape` should be called with `arg1.i[0]` containing the new buffer size in bytes. The device driver will clip this value to its internal minimum and maximum buffer size limits.

C Syntax Example

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

gescape_arg arg1, arg2;

/* A large buffer is good for 1000 line polylines */

arg1.i[0] = 4096;
gescape(fildes,SET_BUFFER_SIZE,&arg1,&arg2);
```

FORTRAN77 Syntax Example

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

arg1(1) = 4096
call gescape(fildes,SET_BUFFER_SIZE,arg1,arg2)
```

Pascal Syntax Example

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

var
    arg1, arg2 : gescape_arg;

begin
    arg1.i[1] := 4096;
    gescape(fildes,SET_BUFFER_SIZE,arg1,arg2);
```

SET_REPLACEMENT_RULE

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

The HP 98705 device supports a replacement rule for each plane (see `drawing_mode` for a description of replacement rules). The HP 98705 hardware uses these replacement rules while doing bit-per-pixel writes. This gescape allows these replacement rules to be set, and enables or disables their use in bit-per-pixel block writes.

Source, Destination, and Pattern values provide the three operands for the replacement rule. The default replacement rule is `SOURCE`, rule number `0x33`. See the tables in the `PATTERN_FILL gescape` for a a more detailed description of three-operand replacement rules.

`arg1.i[0]` is the three-operand (8-bit) replacement rule.

A

`arg1.i[1]` is an integer mask indicating the plane(s) to which this replacement rule corresponds.

`arg1.i[2]` is an enable flag indicating whether or not the replacement rules should be used in bit-per-pixel block writes.

Following are the legal values for `arg1.i[2]` and their meaning.

`arg1.i[2] = 0`	Turn replacement rule mode off (ignores other parameters)
`arg1.i[2] = 1`	Turn replacement rule mode on (uses other parameters)
`arg1.i[2] = 2`	Leave replacement rule mode as it is (on or off) but sets values based on other parameters.

`arg2` is ignored.

This `gescape` is useful for two color bit-per-pixel block writes. Being able to set a replacement rule per plane allows you to select any entry in the color table when all ones or all zeros are written into the planes as a result of a bit-per-pixel write. (Note that `R_BIT_MODE` must be called to enable bit mode before bit-per-pixel mode will work.) Unlike `R_BIT_MODE`, which will only write to one plane (set with `R_BIT_MASK`), the `SET_REPLACEMENT_RULE gescape` causes a "one" bit in the source to be expanded into a "one" written to each plane using the rule set for that plane. The planes actually written will depend on which are enabled. (see `write_enable` for details).

Note If the overlay planes were opened as as three plane device, setting a replacement rule for the fourth plane will have no effect.

Note that these replacement rules are only used when doing bit-per-pixel block writes, and when enabled as described above. These replacement rules are entirely independent of the drawing_mode replacement rule.

The following example writes a bit-per-pixel source image in array bitdata to the frame buffer with one bits expanding to color index 5, and zero bits expanding to color 6, using an effective rule of SOURCE.

C Syntax Example

```
#include ⟨starbase.c.h⟩

        static unsigned int bitdata[ ]={
                0xffffffff,
                0xa0000009,
                0x88000021,
                0x82000081,
                0x80800201,
                0x80200801,
                0x80082001,
                0x80028001,
                0x80028001,
                0x80082001,
                0x80200801,
                0x80800201,
                0x82000081,
                0x88000021,
                0xa0000009,
                0xffffffff
        };

        /*  Type gescape_arg is defined via typedef in starbase.c.h. */

        gescape_arg arg1, arg2;
        int rr00, rr01, rr10, rr11;
        int fg, bg;


        /*  This call enables bit-per-pixel read and write. */
        arg1.i[0] = 1;
        gescape(fildes, R_BIT_MODE, &arg1,&arg2);

        /*
        The following four replacement rules presume an effective
        rule of SOURCE.  Rule rr[m][n] is the rule to use when the
        foreground color in a particular plane requires a bit value
        of m, and the background color requires a value of n.
*/
rr00 = 0x00;
rr01 = 0xCC;
rr10 = 0x33;
rr11 = 0xff;

fg=5;
bg=6;
arg1.i[2] = 1;

/*
        The planes that receive each rule are determined by
        the bit-wise intersections of the foreground and background
        colors.
*/
arg1.i[0] = rr00;
arg1.i[1] = ~(fg) & ~(bg);
gescape(fildes, SET_REPLACEMENT_RULE, &arg1,&arg2);
arg1.i[0] = rr01;
arg1.i[1] = ~(fg) & bg;
gescape(fildes, SET_REPLACEMENT_RULE, &arg1,&arg2);
arg1.i[0] = rr10;
arg1.i[1] = fg & ~(bg);
gescape(fildes, SET_REPLACEMENT_RULE, &arg1,&arg2);
arg1.i[0] = rr11;
arg1.i[1] = fg & bg;
gescape(fildes, SET_REPLACEMENT_RULE, &arg1,&arg2);

/*
        Here is the actual block write to produce the
        two-color image.
*/
dcblock_write (fildes, 128, 128, 32, 16, bitdata, TRUE);
```

FORTRAN77 Syntax Example

```
      integer*4 arg1(64), arg2(64)
      integer*4 rr00, rr01, rr10, rr11
      integer*4 fg, bg
      integer*4 bitdata(16)
      data bitdata/Z'FFFFFFFF',
     +            Z'A0000009',
     +            Z'88000021',
     +            Z'82000081',
     +            Z'80800201',
     +            Z'80200801',
     +            Z'80082001',
     +            Z'80028001',
     +            Z'80028001',
     +            Z'80082001',
     +            Z'80200801',
     +            Z'80800201',
     +            Z'82000081',
     +            Z'88000021',
     +            Z'A0000009',
     +            Z'FFFFFFFF'/


C     This call enables bit-per-pixel read and write.
      arg1(1)=1
      call gescape(fildes, R_BIT_MODE, arg1,arg2)

C     The following four replacement rules presume an effective
C     rule of SOURCE.  Rule rr[m][n] is the rule to use when the
C     foreground color in a particular plane requires a bit value
C     of m, and the background color requires a value of n.
      rr00=Z'00'
      rr01=Z'CC'
      rr10=Z'33'
      rr11=Z'FF'

      fg=5
      bg=6
      arg1(3)=1

C     The planes that receive each rule are determined
C     by the bit-wise intersections of the foreground and
C     background colors.
      arg1(1)=rr00
      arg1(2)=(.NOT.(fg)) .AND. (.NOT.(bg))
      call gescape(fildes, SET_REPLACEMENT_RULE, arg1,arg2)
      arg1(1)=rr01
      arg1(2)=(.NOT.(fg)) .AND. bg
      call gescape(fildes, SET_REPLACEMENT_RULE, arg1,arg2)
      arg1(1)=rr10
      arg1(2)=fg .AND. (.NOT.(bg))
      call gescape(fildes, SET_REPLACEMENT_RULE, arg1,arg2)
      arg1(1)=rr11
      arg1(2)=fg .AND. bg
      call gescape(fildes, SET_REPLACEMENT_RULE, arg1,arg2)

C     Here is the actual block write to produce the
C     two-color image.
      call dcblock_write (fildes, 128, 128, 32, 16, bitdata, TRUE)
```

Pascal Syntax Example

```
type
        bitarray = record
                case integer of
                        0:  (i: packed array [0..15] of integer);
                        1:  (b: packed array [0..63] of gbyte);
                end;

var
        {  Type gescape_arg is defined via typedef in starbase.p1.h. }

        arg1, arg2: gescape_arg;
        rr00, rr01, rr10, rr11: integer;
        mask00, mask01, mask10, mask11, plane: integer;
        fg_plane, bg_plane: boolean;
        fg, bg: integer;
        bitdata: bitarray;


begin
        bitdata.i[0]  := hex('FFFFFFFF');
        bitdata.i[1]  := hex('A0000009');
        bitdata.i[2]  := hex('88000021');
        bitdata.i[3]  := hex('82000081');
        bitdata.i[4]  := hex('80800201');
        bitdata.i[5]  := hex('80200801');
        bitdata.i[6]  := hex('80082001');
        bitdata.i[7]  := hex('80028001');
        bitdata.i[8]  := hex('80028001');
        bitdata.i[9]  := hex('80082001');
        bitdata.i[10] := hex('80200801');
        bitdata.i[11] := hex('80800201');
        bitdata.i[12] := hex('82000081');
        bitdata.i[13] := hex('88000021');
        bitdata.i[14] := hex('A0000009');
        bitdata.i[15] := hex('FFFFFFFF');
```

```
{  This call enables bit-per-pixel read and write. }
arg1.i[1] := 1;
gescape(fildes, R_BIT_MODE, arg1,arg2);

{
        The following four replacement rules presume an effective
        rule of SOURCE.  Rule rr[m][n] is the rule to use when the
        foreground color in a particular plane requires a bit value
        of m, and the background color requires a value of n.
}
rr00 := hex('00');
rr01 := hex('CC');
rr10 := hex('33');
rr11 := hex('ff');

fg:=5;
bg:=6;
arg1.i[3] := 1;

{
        The planes that receive each rule are determined by the
        bit-wise intersections of the foreground and background
        colors.  Because Pascal does not support bitwise masking
        operations, the masks are built up in a loop.
}
mask00 := 0;
mask01 := 0;
mask10 := 0;
mask11 := 0;

plane:=128;
while plane > 0 do
begin
        fg_plane := (plane <= fg);
        bg_plane := (plane <= bg);
        if (not fg_plane) and (not bg_plane)
                then mask00 := mask00 + plane
        else if (not fg_plane) and bg_plane
                then mask01 := mask01 + plane
        else if fg_plane and (not bg_plane)
                then mask10 := mask10 + plane
        else
                mask11 := mask11 + plane;
        if fg_plane then fg := fg - plane;
        if bg_plane then bg := bg - plane;

        plane := plane div 2;
```

```
end;

arg1.i[1] := rr00;
arg1.i[2] := mask00;
gescape(fildes, SET_REPLACEMENT_RULE, arg1,arg2);
arg1.i[1] := rr01;
arg1.i[2] := mask01;
gescape(fildes, SET_REPLACEMENT_RULE, arg1,arg2);
arg1.i[1] := rr10;
arg1.i[2] := mask10;
gescape(fildes, SET_REPLACEMENT_RULE, arg1,arg2);
arg1.i[1] := rr11;
arg1.i[2] := mask11;
gescape(fildes, SET_REPLACEMENT_RULE, arg1,arg2);

{
        Here is the actual block write to produce the
        two-color image.
}

dcblock_write (fildes, 128, 32, 16, bitdata.b, 1);
```

SMD_ALLOCATE_MEMORY

The ⟨*op*⟩ parameter is `SMD_ALLOCATE_MEMORY` (supported by all SMD drivers).

This `gescape` forces allocation of the memory buffer. The memory buffer is allocated according to the current X, Y and depth definitions. The `gescape` examines the current X, Y dimensions and the current depth to determine the size of the buffer allocation. If the memory buffer has already been allocated, this `gescape` determines if a new, larger buffer is required.

Syntax

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

`arg2.i` contains the following return information:

- `arg2.i[0]` is success or failure. Failure occurs if the memory buffer cannot be allocated.
- `arg2.i[1]` is the current pointer to the frame buffer.
- If `arg2.i[0]` indicates success, `arg2.i[2]` is the number of bytes allocated for the memory buffer. If `arg2.i[0]` indicates failure, `arg2.i[2]` is the number of bytes that are available.

SMD_DEFINE_DEPTH

The ⟨*op*⟩ parameter is `SMD_DEFINE_DEPTH` (supported by `SMDpixel` and `SMDplane` only).

This `gescape` call allows definition of the logical depth (number of planes) when using the `SMDpixel` (the physical depth always remains 8) packing format and the **physical** depth for `SMDplane`. If you called `gopen` with `SMDpixel` driver, the valid logical depth values are 1, 3, 4, 6, or 8. The physical depth values for `SMDplane` are 1, 3, 4, 6, 8, 16, or 24.

Changing the depth of the frame buffer changes the size of the color table. The color table size is equal to 2 raised to the number of frame buffer planes up to 256 entries. For example, $2^8 = 256$. This `gescape` always causes the color table to be reinitialized to the Starbase default values.

The SMD treats the color table assuming that the resulting device used to display the memory buffer has a hardware color map. This means that when the SMD gets an index value for the color of a primitive, it uses this index for writing into the frame buffer. This is different from drivers for monochromatic displays (the `hp300h` and `hp300l` drivers[4]).

This `gescape` call can occur at any time, but if the memory buffer has not been allocated, this `gescape` will not allocate the memory buffer. The memory buffer is allocated in the following situations:

- Graphics primitives are done to the memory buffer,
- `SMD_ALLOCATE_MEMORY` `gescape` is called, or
- `R_GET_FRAME_BUFFER` is called.

[4] The `hp300h` and `hp300l` drivers look at the color map definition for the index provided from Starbase and determine if that index represents "color" or "no color." If it represents color, the driver uses a pen value of one. If it represents no color, the driver uses a pen value of zero. For example, application drawing to a monochromatic `300h` changes the color table definition such that index 0 is white and index 1 is black. It specifies `line_color_index` with index 0. The driver does not write index 0 values into the frame buffer. Instead, it determines that the color at index 0 in the color table is white and writes index value 1 into the frame buffer because monochromatic devices do not have a hardware color map.

If the memory buffer has already been allocated, the current memory buffer is not altered by this `gescape`; however, subsequent graphics primitives are done with the new range of color indexes.

Syntax

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

`arg1.i[0]` contains the number of planes.

`arg2.i` returns the following information:

- `arg2.i[0]` is success or failure. Failure can occur if you specify an invalid depth value.
- `arg2.i[1]` is the current frame buffer pointer.
- `arg2.i[2]` is the number of bytes currently required by the frame buffer.

If failure is indicated, the application must call `inquire_gerror` to know the complete nature of the error. If the error was 6 (`IMPROPER_VALUE`), you passed in an improper depth value.

If the depth definition was for an invalid depth value,

- `arg2.i[0]` is returned indicating failure,
- `arg2.i[1]` contains the current memory buffer pointer (this pointer is `NULL` if the memory buffer has not yet been allocated), and
- `arg2.i[2]` is the number of bytes required for the memory buffer.

SMD_DEFINE_XY

The ⟨*op*⟩ parameter is `SMD_DEFINE_XY` (supported by all SDM drivers).

This `gescape` lets you define the X, Y dimensions of the memory buffer. The memory buffer still uses the packing format indicated in the `gopen` call. The redefinition can occur at any time.

If a memory buffer is currently defined (not a `NULL` pointer), and the redefinition requires a larger buffer, the current buffer is deallocated, and a new buffer is allocated. Graphics primitives you may have done are not retained. This `gescape` works on memory buffers supplied by you (see "User-Supplied Memory Buffers"), but you should be aware that if your new X, Y definition requires reallocation of the buffer, your supplied memory is deallocated and a new memory region is allocated instead. Thus, the local copy of the pointer to the supplied memory region is no longer valid.

If the memory buffer has not yet been allocated or if the memory buffer is currently undefined, the memory buffer pointer is `NULL`. This `gescape` does not allocate the memory buffer. The memory buffer is allocated in the following situations:

- Graphics primitives are being done to the memory buffer,
- `SMD_ALLOCATE_MEMORY` `gescape` is called, or
- `R_GET_FRAME_BUFFER` `gescape` is called.

The X and Y indexes in a frame buffer each have to be $2^{15}-1$ (32,767) or less. This is because the vector generation algorithms can only handle up to 15 bits of addressing along each axis.

The maximum size of a frame buffer is discussed further in "The Starbase Memory Driver" in the *Starbase Graphics Techniques* manual.

By redefining the size, the Virtual Device Coordinate to Device Coordinate (VDC-to-DC) mapping is recomputed. The current VDC extent definition remains the same; however, P1 and P2 are set back to `FRACTIONAL` 0, 0, 0 to 1, 1, 1. Redefinition of the memory buffer causes the memory buffer to be cleared to the background color if you opened with `mode` containing `INIT` or `RESET_DEVICE`.

Syntax

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

arg1 contains two integer values, maximum X and maximum Y. If you want a memory buffer 512×512 pixels in size, arg1.i[0]=512 and arg1.i[1]=512. Thus, the maximum DC (X, Y) you can reference is 511, 511.

arg2.i contains the following return information:

- arg2.i[0] is success or failure. Failure can occur if you specify X or Y range values that are invalid; or the X, Y redefinition required a reallocation of the memory buffer, and the amount of memory now being requested cannot be allocated.
- arg2.i[1] is the current pointer to the frame buffer or NULL if no frame buffer has been allocated.
- arg2.i[2] depends on the nature of the failure.
  - If the failure was due to the X or Y size exceeding 65,535, arg2.i[2] is the number of bytes for the frame buffer (based on the previous definitions of X and Y).
  - If the failure was due to being unable to allocate the memory based on the new X, Y size, arg2.i[2] contains the size (in bytes) of the buffer which could have been allocated had you requested that size.

The application must do an inquire_gerror if failure is indicated to know the complete nature of the error. If the error number is 2049, Starbase was unable to allocate the memory buffer. If the error number is 6 (IMPROPER_VALUE), then the X and/or Y values were invalid.

If arg2.i[0] returns indicating failure, a Starbase error is also issued to stderr.

If the device is gopen ed with the INIT or RESET_DEVICE bits set, the memory buffer is allocated at this time. If you redefine the X, Y size too large to be allocated,

- failure is returned to you in arg2.i[0], and
- arg2.i[1] is the NULL pointer.

This happens because the SMD deallocates the first buffer before attempting to allocate the second, larger one. But since the SMD is unable to allocate the required amount of memory, arg2.i[1] is returned with the NULL pointer.

If your application then redefines the X, Y size to something smaller (since the current memory buffer pointer is now `NULL`), the SMD does not allocate the memory at the time of that `SMD_DEFINE_XY` `gescape` call. SMD returns

- success in `arg2.i{0}`(provided the X and Y dimension values were valid),
- a `NULL` pointer in `arg2.i[1]`, and
- the number of bytes required in `arg2.i[2]`.

At this point, you should call `gescape` with `SMD_ALLOCATE_MEMORY` to force allocation of the memory buffer. This insures that the SMD can allocate the required memory. You could choose not to allocate the memory through `SMD_ALLOCATE_MEMORY` letting the memory buffer be allocated at the time of the first graphics primitive. However, if the X, Y is still such that the SMD cannot allocate the memory, a Starbase error is generated at the time of the graphics primitive, and you have no way of knowing how many bytes are available at allocation time.

SMD_GET_MEM_REQUIRED

The $\langle op \rangle$ parameter is `SMD_GET_MEM_REQUIRED` (supported by all SMD drivers).

This `gescape` determines the amount of memory required for a memory buffer based on the current packing format and X, Y dimensions. Use this value to `malloc` the memory to be supplied to the SMD in the `SMD_SUPPLY_MEM_BUFF` `gescape` call.

Syntax

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

`arg2.i[0]` is returned with the number of bytes required for the memory buffer.

SMD_SUPPLY_MEM_BUFF

The ⟨*op*⟩ parameter is `SMD_SUPPLY_MEM_BUFFU` (supported by all SMD drivers).

This `gescape` allows you to pass a pointer to a block of memory for the SMD to use as its memory buffer.

The SMD, by default, allocates its own memory buffer at `gopen` time if you open with ⟨*mode*⟩ equal to `INIT` or `RESET_DEVICE`. If you open with ⟨*mode*⟩=0, the buffer is not allocated, and Starbase expects either

- allocating the buffer by supplying a pointer to the buffer, or
- a `gescape` to force allocation of the buffer (see "Allocate a Frame Buffer" or "Get Frame Buffer Pointer").

It is your responsibility to understand the format of the memory buffer and the amount of memory required. (See "Determining Memory Requirements") If you do not allocate enough memory, a system error may occur when the SMD tries to write beyond the memory area.

When supplying a memory buffer that is not going to use the default X, Y (dimensions and depth), you should define X, Y dimensions via `SMD_DEFINE_XY` and `SMD_DEFINE_DEPTH` respectively before supplying the memory buffer to the SMD. Otherwise, the SMD assumes the frame buffer X, Y size and depth as the defaults and set up its VDC-to-DC transformation accordingly.

The user-supplied buffer is not initialized to the background color by the driver.

Syntax

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

`arg1.i[0]` contains the pointer to the memory buffer to be used by the SMD.

`arg2.i[0]` returns success or failure.

`arg2.i[1]` returns the current pointer to the memory buffer.

`arg2.i[2]` returns the number of bytes currently required by the frame buffer.

An `inquire_gerror` call should be made to determine the exact nature of the error. If the error was 11 (`NULL_PTR`), you passed in a `NULL` pointer. If the error was 6 (`IMPROPER_VALUE`), you tried to supply a memory buffer after the memory buffer had already been allocated.

STEREO

Important The `STEREO` gescape switches the HP graphics subsystem to stereo output mode (described below). You are responsible for generating output for left and right eye images. Display of stereo images requires the use of third party hardware. For a list of vendors of stereo-capable equipment, contact your local HP sales office.

Note True stereo images can only be created in raw mode. The X Windows environment does not support stereo output.

The ⟨*op*⟩ parameter is `STEREO`.

A

This function is provided to support the use of a stereoscopic display system. It requires the use of a third party, external display mechanism (monitor and active shutter) to provide a time-multiplexed stereoscopic display. In order to support synchronization of the external shutter with the video cycle, a timing signal output is provided on the display board (on a BNC connector next to the RGB output).

The `STEREO` function places the video display in a 1280 x 512 (120 Hz) output mode. In this mode the displayed frame is alternately sourced from the upper-left and the lower-left 1280 x 512 areas of the frame buffer.

These two areas constitute the stereo-pair, with the upper buffer providing the left image and the lower buffer providing the right image.

In this mode the vertical retrace of each frame is setup to the full vertical sweep of the monitor. As a result the 1280 x 512 frame fills the entire monitor area. This means that each frame buffer pixel is displayed at a 1 to 2 aspect ratio, being twice as tall as wide.

This `gescape` can be used with the HP 98735, HP 98736, HP 98765, HP 98766, CRX-24 and CRX-48Z device drivers to control the display hardware.

To activate the stereo display mode the gescape should be called with: `arg1.i[0]=1`.

Setting `arg1.i[0]=0` will deactivate the stereo display mode and return the display to its normal 1280x1024 mode.

The following examples activate the stereo function with the first `gescape` call and then deactivate it with the second `gescape` call.

C Syntax Example

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

gescape_arg arg1, arg2;
    :
arg1.i[0]=1;
gescape(fildes,STEREO,&arg1,&arg2); /* ON */
arg1.i[0]=0;
gescape(fildes,STEREO,&arg1,&arg2); /* OFF */
```

FORTRAN77 Syntax Example

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

Pascal Syntax Example

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

var
   arg1,arg2:gescape_arg;
>       :
begin
   arg1.i[1] := 1;
   gescape(fildes,STEREO,arg1,arg2); { ON }
   arg1.i[1] := 0;
   gescape(fildes,STEREO,arg1,arg2); { OFF }
```

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.

See the `R_GET_WINDOW_INFO` `gescape` for a C program example.

The following examples switch semaphore operations on.

C Syntax Example

```
/* 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 Example

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

Pascal Syntax Example

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

var
   arg1,arg2 : gescape_arg;

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

TC_HALF_PIXEL

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

This `gescape` allows access to half pixels during `block_read` and `block_write` procedures. Each time it is called, the definition is changed to the other possibility. Initially, it is 1 byte per pixel. After the first call, it is 2 bytes per pixel. The second call returns it to 1 byte per pixel, etc.

This `gescape` will allow more detailed raster operations. When 2-bytes per pixel is enabled, a `block_read` or `block_write` call must pass a pointer to a storage area sufficient for the operation. Each row will occupy 2*dx bytes. So the storage required is dy*2*dx bytes.

The `arg1` and `arg2` parameters are ignored.

C Syntax Example

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

FORTRAN77 Syntax Example

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

Pascal Syntax Example

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

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

TEXTURE_CONTROL

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

This `gescape` allows the selection of one of three texture map filters and offset values to be used for RIP filtered maps. The `gescape` takes 3 floating point numbers.

The offset values used in s and t respectively are specified in `arg1.f[1]` and `arg2.f[2]` and are used exclusively for RIP maps. These values are used as fine controls for adjusting when Starbase switches from one map to the next RIP map in s or t. These are base 2 numbers, so values should be chosen of the form 2.0, 4.0, 8.0, ... , 1024. Values larger than 1.0 force the switch between maps to happen earlier, values less than 1.0 will slow the switch between maps. The use of these parameters is particularly effective for animated texture maps that consist of high frequency data.

The filters available are specified through `arg1.i[0]` and are:

`POINT MAPS`	This texture mapping method is done by computing (s,t) values at the center of each pixel. The (s,t) values are then used to find a single point in the texture map. When adjacent pixels take large steps in texture space, aliasing of texture maps becomes a problem in this mode. To get this option the `gescape` should be called with: `arg1.f[0]=0.0`
`RIP MAPS`	This is the default method. The aliasing problem mentioned in the above texture mapping method is reduced by filtering the texture map in s and t. The filtering is done by a image pyramid scheme where the original texture map is recursively filtered along each axis independently. This technique is referred to as a RIP map and takes up four times the off screen memory as do `POINT MAPS`. To get this option the gescape should be called with: `arg1.f[0]=1.0`
`PRE-FILTERED RIP`	This option allows you to define a RIP texture map that has been prefiltered. With this mode, you can use your own filtering algorithm to make the RIP map, or in conjunction

with the `TEXTURE_RETRIEVE` `gescape`, load a texture map into off screen that has already been previously filtered by Starbase. This mode can be used to improve the performance of texture mapping. For this mode, you must specify the s and t size in `define_texture` as the physical size of the RIP map (twice the original size in s and t).

To get this option the gescape should be called with: `arg1.f[0]=2.0`

A change in the texture mapping mode will remove all textures from off screen and reset the current texture index and back facing texture index to 0.

The following examples select RIP texture mapping and cause switching between RIP maps to happen earlier than normal.

C Syntax Example

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

gescape_arg arg1, arg2;
    ⋮
arg1.f[0]=1.0;
arg1.f[1]=2.0;
arg1.f[2]=2.0;
gescape(fildes,TEXTURE_CONTROL,&arg1,&arg2);
```

FORTRAN77 Syntax Example

```
real arg1(64),arg2(64)
arg1(1)=1.0
arg1(2)=2.0
arg1(3)=2.0
call gescape(fildes,TEXTURE_CONTROL,arg1,arg2)
```

Pascal Syntax Example

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

var
   arg1,arg2:gescape_arg;
        ⋮
begin
   arg1.f[1] := 1.0;
   arg1.f[2] := 2.0;
   arg1.f[3] := 2.0;
   gescape(fildes,TEXTURE_CONTROL,arg1,arg2);
```

TEXTURE_DOWNSAMPLE

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

This `gescape` provides a means for explicitly down sampling a texture map. The `gescape` takes 3 integers as parameters. `arg1.i[0]` is the index of the texture to be down sampled. The texture should already be defined via `define_texture`, if it is not an error is returned. `arg1.i[1]` is the factor by which the texture map should be down sampled. A value of 1 for this parameter means no down sampling should be done, a value of 2 means that the original texture should be down sampled once, a value of 3 should down sample twice, etc. Finally, `arg1.i[2]` indicates whether this is to be a front facing texture map (0) or a back facing texture map(1).

`arg2.i[0]` returns the factor by which the texture was down sampled. This is the same behavior as the value returned by `texture_index`. The texture may be down sampled more times than you request if not enough off screen memory exists to fulfill your original request.

The following examples request the texture at index 3 to be down sampled once as a back face texture map. It is assumed that the texture for this index is already defined.

C Syntax Example

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

gescape_arg arg1, arg2;
    ⋮
arg1.i[0]=3;
arg1.i[1]=2;
arg1.i[2]=1;
gescape(fildes,TEXTURE_DOWNSAMPLE,&arg1,&arg2);
```

FORTRAN77 Syntax Example

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

Pascal Syntax Example

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

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

TEXTURE_RETRIEVE

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

This `gescape` allows you to retrieve down sampled and filtered texture maps from off screen. In conjunction with the `PRE-FILTERED RIP` mode of the `TEXTURE_CONTROL` `gescape`, this `gescape` provides a means of quickly writing textures to off screen by eliminating the down sampling and filtering steps. The `gescape` also provides a means of determining the size of the off screen texture area used so that you may allocate sufficient memory to the hold the returned map. The `gescape` takes 3 integers as parameters.

'arg1.i[0]'' is the index of the texture to be retrieved. The texture should already be in off screen, if it is not an error is returned.

A

`arg1.i[1]` has a value of 1 if a texture is to be retrieved from off screen. In this case, `arg1.i[2]` is an integer pointer to memory allocated by you to hold the off screen texture. If `arg1.i[1]` is 0, then no retrieval of the texture map occurs and only the off screen s and t sizes of the texture area are returned. In this case, you can determine the size of the off screen area used by the texture so that the correct amount of memory can be allocated to retrieve the texture.

The returned value in `arg2.i[0]` is the off screen s size and the off screen t size is returned in `arg2.i[1]`. The memory pointed to by the character pointer passed in via `arg1.i[2]` holds the retrieved texture when `arg2.i[1]` is 1.

The following examples place the off screen texture at index 64 into the array `texture`. It is assumed that the array `texture` is previously declared to be of sufficient size to hold the off screen texture.

C Syntax Example

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

gescape_arg arg1, arg2;
    ⋮
arg1.i[0]=64;
arg1.i[1]=1;
arg1.i[2]=(int)texture;
gescape(fildes,LS_OVERFLOW_CONTROL,&arg1,&arg2);
```

FORTRAN77 Syntax Example

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

Pascal Syntax Example

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

var
   arg1,arg2:gescape_arg;
        :
begin
   arg1.i[1] := 64;
   arg1.i[2] := 1;
   arg1.i[3] := texture;
   gescape(fildes,LS_OVERFLOW_CONTROL,arg1,arg2);
```

TOGGLE_2D_COLORMAP

The ⟨*op*⟩ parameter is `TOGGLE_2D_COLORMAP`.

This `gescape` enables or disables a display mode which interprets the color map as a two dimensional color map. This display modes provides the capability to interpolate two individual 6 bit indices at one time. This display mode uses the 4096 entry color map which is assumed to be initialized by you. To allow for double buffering and the correct data manipulation, you must have entered 12 bit indexing in CMAP_MONOTONIC shade mode prior to enabling this two dimensional color map `gescape`.

If enabled, this mode interpolates the red and green values of all primitives over 6 bits each. The blue values are ignored. The resulting two 6 bit indices are then combined to result in a 12 bit index by adding the 6 bit red index multiplied by 64 to the 6 bit green index. This 12 bit index is used to look up an rgb color in the 4096 entry color map.

In this mode, the color map can be visualized as two dimensional in that it is broken up into 64 color buckets with each bucket having 64 color entries. With this viewpoint, the red data selects the color bucket and the green data selects the color within that bucket.

`arg1.i[0]` is used to enable (1) or disable (0) the two dimensional color map mode.

`arg1.i[1]` is used to enable (1) or disable (0) processing filled polygons through light source equations. This mode can not be turned on unless the device is in `MODEL_XFORM` mode.

If the two dimensional color map mode is enabled, `arg2.i[0]` returns 1, otherwise it returns 0.

If the current display mode is not 12 bit indexing in CMAP_MONOTONIC shade mode then an error is returned and the two dimensional color map mode is not enabled.

The following examples enable the two dimensional color map mode. It is assumed that the device is already in 12 bit indexing CMAP_MONOTONIC shade mode. Processing filled polygons is not turned on.

C Syntax Example

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

gescape_arg arg1, arg2;
   ⋮
arg1.i[0]=1;
arg1.i[1]=0;
gescape(fildes,TOGGLE_2D_COLORMAP,&arg1,&arg2);
```

FORTRAN77 Syntax Example

```
integer*4 arg1(64),arg2(64)
arg1(1)=1
arg1(2)=0
call gescape(fildes,TOGGLE_2D_COLORMAP,arg1,arg2)
```

Pascal Syntax Example

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

var
   arg1,arg2:gescape_arg;
       ⋮
begin
   arg1.i[1] := 1;
   arg1.i[2] := 0;
   gescape(fildes,TOGGLE_2D_COLORMAP,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.

The arg1 parameter contains the mask.

The arg2 parameter is ignored.

Table A-10.

15	...	2	1	0

Table A-11.

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 Example

```
gescape_arg arg1, arg2; /*typedef defined in starbase.c.h */

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 Example

```
        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 Example

```
{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);
```

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);
```

ZBANK_ACCESS

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

This gescape controls access to the Z-buffer through the `block_read` and `block_write` functions. If `arg1.i[0] == TRUE`, Z-buffer access is enabled. If `arg1.i[0] == FALSE`, Z-buffer access is disabled.

When Z-buffer access is enabled, `bank_switch` will allow you to select the Z-buffer bank for reading and writing. The `inquire_fb_configuration` function will include the Z-buffer in the `image_banks` parameter.

When Z-buffer access is disabled, `bank_switch` will report an error when trying to select the Z-buffer. The `inquire_fb_configuration` function will not include the Z-buffer in the `image_banks` parameter.

The default is to have Z-buffer access disabled on those devices that implement this gescape.

C Syntax

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

gescape_arg arg1;

arg1.i[0] = TRUE;         /* Enable Z-buffer access. */
gescape(fildes, ZBANK_ACCESS, arg1, NULL);
```

FORTRAN 77 Syntax

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

arg1(1) = TRUE
call gescape(fildes, ZBANK_ACCESS, arg1, arg2)
```

Pascal Syntax

```
var
  arg1, arg2 : gescape_arg;

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

ZBUFFER_ALLOC

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

The HP 98721 device uses offscreen frame buffer memory for Z-buffering. This `gescape` controls offscreen Z-buffer usage so it is only valid to use on systems which do not have dedicated Z-buffers.

Starbase uses off-screen memory to keep data to make various primitives go as fast as possible. When the HP 98721 driver in opened with the `gopen` command, a strip 128 bytes wide and 1024 lines high in the right-most part of the frame buffer is allocated for Starbase storage. The strip between the left edge of the undisplayed region and the left edge of Starbase storage in each of the frame buffers and any undisplayable banks is allocated to Z-buffer area.

The following table shows the maximum number of pixels that can be rendered in one pass in the default case:

Table A-12. Default Off-Screen Buffer Allocation

Configuration	Resolution	Comments
8 planes	320×1024 pixels	
16 planes	1280×1024pixels	8 planes, single buffer
16 planes	640×1024 pixels	8 planes, double buffer
24 planes	1280×1024 pixels	8 planes, single buffer
24 planes	1280×1024 pixels	8 planes, double buffer
24 planes	960×1024 pixels	24 planes, single buffer
32 planes	1280×1024 pixels	24 planes, single buffer

For configurations shown in the table that occupy less than the full screen, the actual number of pixels that can be rendered in one pass may be less than what is shown in the table. It depends on the actual physical limits of the viewport area. If Z-buffer memory is smaller than the amount needed to render the area within the viewport limits, the primitive data must be sent to Starbase more than once.

This `gescape` also allows you to allocate off-screen memory for other uses or to use the Starbase storage area for Z-buffer.

Pass in `arg1` the number of 128 byte by 1024 line strips in the displayable banks to be used for Z-buffering. The minimum number is 1, the maximum number is 6, and the default is 5.

The `arg2` parameter is ignored.

The following example allocates all of the off-screen memory for Z-buffer (this will wipe out any graphics raster cursors and other raster storage used by this or other drivers).

C Syntax Example

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

gescape_arg arg1, arg2;

arg1.i[0]=6;
gescape(fildes,ZBUFFER_ALLOC,&arg1,&arg2);
```

FORTRAN77 Syntax Example

```
real arg1(64),arg2(64)
arg1(1)=6
call gescape(fildes,ZBUFFER_ALLOC,arg1,arg2)
```

Pascal Syntax Example

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

var
   arg1,arg2:gescape_arg;

begin
   arg1.i[1] := 6;
   gescape(fildes,ZBUFFER_ALLOC,arg1,arg2);
```

Exceptions

The HP 98721 driver uses off screen memory for storing raster cursor information and `FILL_COLOR` in `CMAP_NORMAL` mode. See "Cautions" section of the device driver for more information. Do not use the storage area if either of these functions are used. Other drivers may use this area as well. See the appropriate driver manual.

ZSTATE_RESTORE

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

The HP 98721 device uses offscreen frame buffer memory for Z-buffering. On devices which have a dedicated Z-buffer, offscreen frame buffer memory is not used for Z-buffering. Therefore, this `gescape` is not needed and will have no effect.

The way off-screen memory is allocated depends on:

- the `cmap mode` parameter of shade_mode
- whether double buffering is on or off.
- how many banks of memory are installed.
- viewport area.
- the `ZBUFFER_ALLOC` `gescape` setting.

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. When the HP 98721 driver is opened to the overlay planes and hidden-surface is turned, on the driver will use only the memory in the off-screen part of bank 0 by default. This `gescape` allows you to save a couple of internal variables that specify the current Z-buffer allocation in the image planes then restore that allocation in the overlay planes. Of course this method will only work if there is enough off-screen memory to support a one pass Z-buffer.

To get a three-dimensional cursor effect, this `gescape` must be used in conjunction with another `gescape`. The `gescape` `ZWRITE_ENABLE` allows the primitives to be drawn using the Z-buffer, but they do not modify the Z-buffer.

The following sequence must be observed for the overlay cursors to work properly:

```
/* open driver in image planes */

fildes=gopen("/dev/crt",OUTDEV,"hp98721",INIT|THREE_D|MODEL_XFORM);

/* setup commands that affect zbuffer allocation */
view_port(fildes,x1,y1,x2,y2);                    /* if desired */
shade_mode(fildes,mode,shade);                    /* if desired */
double_buffer(fildes,on,planes);                   /* if desired */
gescape(fildes,ZBUFFER_ALLOC,&arg1,&arg1);          /* if desired */
pass=hidden_surface(fildes,1,cull);            /* pass must be 1 */
/* zbuffer now setup.  Save state */
gescape(fildes,ZSTATE_SAVE,&arg1,&arg2);          /* arg2 has info */

/* don't forget to clear zbuffer */
zbuffer_switch(fildes,1);
draw_complex_object();
gclose(fildes);

/* open driver in overlay planes */
fildes=gopen("/dev/overlay",OUTDEV,"hp98721",INIT|THREE_D|MODEL_XFORM);

/* Setup commands that effect zbuffer allocation */
view_port(fildes,x1,y1,x2,y2);                    /* exactly as above */
gescape(fildes,ZBUFFER_ALLOC,&arg1,&arg1);          /* exactly as above */

/* use following for double bffered cursors, otherwise not necessary */
double_buffer(fildes,TRUE|INIT,1);

/* set the bactground to see thru to image planes */
/* must happen after double buffer for correct effect */
arg1[0]=0;
gescape(fildes,R_TRANSPARENCY_INDEX,&arg1,&arg1);

hidden_surface(fildes,1,cull);
```

```
/* zbuffer now setup.  Now restore state */
/* must come after hidden surface turned on */
gescape(fildes,ZSTATE_RESTORE,&arg2,&arg2);      /* arg2 has info */

/* disable zbuffer writes. */
arg1[0]=0;
gescape(fildes,ZWRITE_ENABLE,&arg1,&arg1);

draw_cursor();
gclose(fildes);
```

The `gescape` `ZSTATE_SAVE` must occur after `hidden_surface` is turned on. After the call, `arg2` contains two integers: `arg2[0]` contains a bit mask of the banks used in the primary Z-buffer, and `arg2[1]` contains the bit mask for the secondary Z-buffer.

The `gescape` `ZSTATE_RESTORE` must occur after `hidden_surface` is turned on. The `arg1` parameter should contain the two integers that were returned in `arg2` of the `ZSTATE_SAVE` `gescape`.

C Syntax Example

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

gescape_arg arg1, arg2;

gescape(fildes,ZSTATE_SAVE,&arg1,&arg2);
arg1.i[0]=arg2.i[0];
arg1.i[1]=arg2.i[1];
gescape(fildes,ZSTATE_RESTORE,&arg1,&arg2);
```

FORTRAN77 Syntax Example

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

call gescape(fildes,ZSTATE_SAVE,arg1,arg2)
    arg1(1)=arg2(1)
    arg1(2)=arg2(2)
call gescape(fildes,ZSTATE_RESTORE,arg1,arg2)
```

Pascal Syntax Example

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

var
   arg1,arg2:gescape_arg;

begin
   gescape(fildes,ZSTATE_SAVE,arg1,arg2);
   arg1.i[1]:=arg2.i[1];
   arg1.i[2]:=arg2.i[2];
   gescape(fildes,ZSTATE_RESTORE,arg1,arg2);
```

ZSTATE_SAVE

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

The HP 98721 device uses offscreen frame buffer memory for Z-buffering. On devices which have a dedicated Z-buffer, offscreen frame buffer memory is not used for Z-buffering. Therefore, this `gescape` is not needed and will have no effect.

The way off-screen memory is allocated depends on:

- the `cmap mode` parameter of `shade_mode`.
- whether double buffering is on or off.
- how many banks of memory are installed.
- viewport area.
- the `ZBUFFER_ALLOC gescape` setting.

This `gescape` was designed specifically to allow the creation of three-dimensional cursors in the overlay planes. To accomplish this objective, one needs to get a primitive drawn in the overlay planes to use the same Z-buffer used to draw the object in the image planes. When the HP 98721 driver is opened to the overlay planes and `hidden_surface` is turned on, the driver will use only the memory in off-screen part of bank 0 by default. This `gescape` allows you to save a couple of internal variables that specify the current Z-buffer allocation in the image planes then restore that allocation in the overlay planes. Of course this method will only work if there is enough off-screen memory to support a one pass Z-buffer.

To get a three-dimensional cursor effect, this `gescape` must be used in conjunction with another `gescape`. The `gescape ZWRITE_ENABLE` allows the primitives to be drawn using the Z-buffer but, they do not modify the Z-buffer.

The following sequence must be observed for the overlay cursors to work properly:

```
 /* open driver in image planes */
 fildes=gopen("/dev/crt",OUTDEV,"hp98721",INIT|THREE_D|MODEL_XFORM);

 /* setup commands that affect zbuffer allocation */
 view_port(fildes,x1,y1,x2,y2);                  /* if desired */
 shade_mode(fildes,mode,shade);                  /* if desired */
 double_buffer(fildes,on,planes);                /* if desired */
 gescape(fildes,ZBUFFER_ALLOC,&arg1,&arg1);      /* if desired */
 pass=hidden_surface(fildes,1,cull);          /* pass must be 1 */
/* zbuffer now setup.  Save state */
gescape(fildes,ZSTATE_SAVE,&arg1,&arg2);          /* arg2 has info */
```

```
/* don't forget to clear zbuffer */
zbuffer_switch(fildes,1);
draw_complex_object();
gclose(fildes);

/* open driver in overlay planes */
fildes=gopen("/dev/overlay",OUTDEV,"hp98721",INIT|THREE_D|MODEL_XFORM);

/* Setup commands that effect zbuffer allocation */
view_port(fildes,x1,y1,x2,y2);                  /* exactly as above */
gescape(fildes,ZBUFFER_ALLOC,&arg1,&arg1);       /* exactly as above */

/* use following for double bffered cursors, otherwise not necessary */
double_buffer(fildes,TRUE|INIT,1);

/* set the bactground to see thru to image planes */
/* must happen after double buffer for correct effect */
arg1[0]=0;
gescape(fildes,R_TRANSPARENCY_INDEX,&arg1,&arg1);

hidden_surface(fildes,1,cull);

/* zbuffer now setup.  Now restore state */
/* must come after hidden surface turned on */
gescape(fildes,ZSTATE_RESTORE,&arg2,&arg2);      /* arg2 has info */

/* disable zbuffer writes. */
arg1[0]=0;
gescape(fildes,ZWRITE_ENABLE,&arg1,&arg1);

draw_cursor();
gclose(fildes);
```

The gescape ZSTATE_SAVE must occur after hidden-surface is turned on. After the call, arg2 contains two integers: arg2[0] contains a bit mask of the banks used in the primary Z-buffer, and arg2[1] contains the bit mask for the secondary Z-buffer.

The gescape ZSTATE_RESTORE must occur after hidden_surface is turned on, and arg1 should contain the two integers that were returned in arg2 or the ZSTATE_SAVE gescape.

C Syntax Example

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

gescape_arg arg1, arg2;

gescape(fildes,ZSTATE_SAVE,&arg1,&arg2);
arg1.i[0]=arg2.i[0];
arg1.i[1]=arg2.i[1];
gescape(fildes,ZSTATE_RESTORE,&arg1,&arg2);
```

FORTRAN77 Syntax Example

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

call gescape(fildes,ZSTATE_SAVE,arg1,arg2)
    arg1(1)=arg2(1)
    arg1(2)=arg2(2)
call gescape(fildes,ZSTATE_RESTORE,arg1,arg2)
```

Pascal Syntax Example

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

var
   arg1,arg2:gescape_arg;

begin
   gescape(fildes,ZSTATE_SAVE,arg1,arg2);
   arg1.i[1]:=arg2.i[1];
   arg1.i[2]:=arg2.i[2];
   gescape(fildes,ZSTATE_RESTORE,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 Example

```
/* 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 Example

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

arg1(1)=0
call gescape(fildes,ZWRITE_ENABLE,arg1,arg2)
```

Pascal Syntax Example

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

var
   arg1,arg2:gescape_arg;

begin
   arg1.i[1]:=0
   gescape(fildes,ZWRITE_ENABLE,arg1,arg2);
```

Index