Starbase Device Drivers Library Manual

Volume 1

HP 9000 Series 300/800 Computers

HP Part Number 98592-90018

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

Notices

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

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

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

Printing History

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

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

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

Contents

INTRO-COMP

Introduction and Device Comparison

This manual documents the Starbase device drivers for the HP 9000 Series 300 and Series 800 computers. The manual assumes that you understand the Starbase graphics library and have access to the following documents:

- *Starbase Graphics Techniques*
- *Starbase Reference*
- Manuals provided with your graphics devices

Organization

Introduction Chapter

The "Introduction and Device Comparison" has a number of tables to help you find and compare driver information. They contain contain the following information:

- The products that are supported for each device driver.
- The software revisions that effect each driver.
- The revisions that effect each formatter.
- The drivers supported on each Series 300/800 computer.
- The graphics libraries that are supported on each window system.

Manual Organization

This manual contains an introduction section and a section for each device driver and formatter. The appendix contains the graphics escape (`gescape`) calls that are used in two or more drivers.

For each device, the following information is provided:

- Device Description—Key device features relative to using Starbase.
- Setting up the Device—Hardware and software configuration.
- Device Initialization—The device's default parameters, how to open the device in a program, etc.
- Starbase Functionality—How Starbase works on the device, what graphic escape sequences (`gescapes`) are provided, etc.

Structure of Starbase

Conceptually, Starbase has two levels (device independent and device dependent):

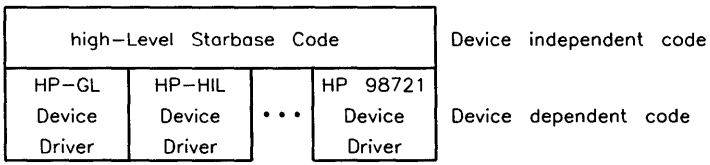


Figure INTRO-COMP-1.

The high-level Starbase code provides user procedures that are device independent. The Starbase device drivers contain the code which drives the graphics devices. Because different devices have different requirements, the drivers are device dependent. Note that a driver (for example, the HP-GL driver) may support several similar devices (for example, the HP 7475, HP 7550 plotters, etc.).

These two levels show the necessity for linking both device-independent code and device-dependent code to generate a usable program.

The absolute path name for the HP Terminal Driver file is:

```
/usr/lib/libddhpterm.a
```

The same driver can be specified using the `-l` format with:

```
-lddhpterm
```

If we were to compile the C program `example.c` to run on an HP 2627 terminal using the "`-l`" format, the following command line would be appropriate:

```
cc example.c -lddhpterm -lsb1 -lsb2
```

It is necessary to use this order of parameters.

The files `sb1` and `sb2` are the high-level Starbase code.

The first table provides a reference list of supported products and device information to be used with them.

Table INTRO-COMP-1. Product Support Information

Device Type	Supported Product	Device Path	Device Driver	Run-Time Library	300	800
HP-GL	HP 7440A HP 7470A HP 7475A HP 7550A HP 7570A HP 7575A HP 7576A HP 7580A/B HP 7585B HP 7586B HP 7595A HP 7596A HP 9111A HP C1600A[12] HP C1601A[12]	`/dev/hpgl`	`hpgl`[1] or `hpgls`[2]	`libddhpgl.a` and `libdvio.a`	X	X

Table INTRO-COMP-1. Product Support Information Continued

Device Type	Supported Product	Device Path	Device Driver	Run-Time Library	300	800
HP-GL CADplt	HP 7510A HP 7550A HP 7570A HP 7575A HP 7576A HP 7580B[6] HP 7585B[6] HP 7586B HP 7595A HP 7596A HP C1600A[12] HP C1601A[12]	/dev/hpgl	CADplt	libddCADplt.a	X	X
HP-GL/2 CADplt2	HP 7595B HP 7596B HP 7599A HP C1600A HP C1601A HP C1602A[11] HP C1620A HP C1625A HP C1627A HP C1629A HP C1631A	/dev/hpgl2	CADplt2	libddCADplt.a	X	X
HP-CGM			hpcgm	libddhpcgm.a	X	X

Table INTRO-COMP-1. Product Support Information Continued

Device Type	Supported Product	Device Path	Device Driver	Run-Time Library	300	800
HP-HIL Buttons	HP 46086A	/dev/hilx or /dev/hilx_x	hp-hil	libddhil.a	X	X
HP-HIL Keyboard	HP 46020A HP 46021A	/dev/hilx or /dev/hilx_x	hp-hil	libddhil.a	X	X
HP-HIL Knobs	HP 46083A HP 46084A	/dev/hilx or/dev/hilx_x	hp-hil	libddhil.a	X	X
HP-HIL Mouse	HP 46060A HP 46060B HP 46095A with HP 46094A	/dev/hilx or /dev/hilx_x	hp-hil	libddhil.a	X	X
HP-HIL Tablets	HP 45911A HP 46087A HP 46088A	/dev/hilx	hp-hil	libddhil.a	X	X
HP-HIL Trackball	HP 80409A	/dev/hilx or /dev/hilx_x	hp-hil	libddhil.a	X	X
Keyboards	HP46020A HP 46021A HP ASCII Terminals	/dev/tty	kbd	libddkbd.a and libddlkbd.a	X	X
X11 Windows	X11 server	/dev/screen/ ptyfile	sox11	libddsox11.a	X	X
X10 Windows	X10 server	/dev/screen/ ptyfile	Xn	libddXn.a	X	X

Table INTRO-COMP-1. Product Support Information Continued

Device Type	Supported Product	Device Path	Device Driver	Run-Time Library	300	800
HP Graphics Terminals	HP 150A HP 150 II HP 2393A HP 2397A HP 2623A HP 2627A HP 2625A HP 2628A	/dev/tty	hpterm	libddhpterm.a	X	X
HP 300 Hi-Res Display	HP 318M HP 98544 HP 98545 HP 98547 HP 98549 Windows/9000 X11	/dev/crt	hp300h	libdd300h.a	X	
HP 300 Lo-Res Display	HP 98542 HP 98543 HP 310 Windows/9000 Xll	/dev/crt	hp300l	libdd300l.a	X	
HP 98550	HP 319C+ HP 98548 HP 98549 HP 98550 Windows/9000[8] X11	/dev/crt	hp98550	libdd98550.a	X	X[5]
HP 98556	HP 98549 HP 98550 Windows/9000[8] X11	/dev/crt	hp98556	libdd98556.a	X	X[5]
HP 98700[3]	HP 98700 Windows/9000	/dev/crt	hp98700	libdd98700.a	X	

Table INTRO-COMP-1. Product Support Information Continued

Device Type	Supported Product	Device Path	Device Driver	Run-Time Library	300	800
HP 98710	HP 98700 with HP 98710	/dev/crt	hp98710	libdd98710.a	X	
HP 98720[4]	HP 98720 HP 319 SRX X11	/dev/crt	hp98720	libdd98720.a	X	X
HP 98720W[4]	HP 98720 Windows/9000[8] HP 319 SRX	/dev/crt	hp98720w	libdd98720w.a	X	X
HP 98721	HP 98720 with HP 98721 HP 319 SRX	/dev/crt	hp98721	libdd98721.a	X	X
HP 98730[9]	HP 98730 Windows/9000[8] X11	/dev/crt	hp98730	libdd98730.a	X	X[10]
HP 98731	HP 98731 Windows/9000[8] X11	/dev/crt	hp98731	libdd98731.a	X	X[10]

1 HP-GL devices with HP-IB interface.
2 HP-GL devices with RS-232 interface.
3 With or without HP 98710 accelerator.
4 With or without HP 98721 accelerator.
5 Only the HP 98550 Display Board is supported and only on Model 825 and 835 computers.
6 For plotters with serial number 2402 or higher.
7 Only supported on the Model 825 computer.
8 Windows/9000 is only supported on the Series 300 computers.
9 With or without HP 98731 accelerator.
10 Supported only on the Model 825 and 835 computers.
11 With HP-GL/2 plug-in cartridge.
12 Only in emulate mode.

Driver Compatibility with High-Level Starbase

Starbase drivers are developed in concert with a particular release of the high-level Starbase code; thus, compatibility between drivers and the high-level code is assured. In the future, however, new drivers may be released without re-releasing the high-level code. To permit determining high-level Starbase and driver compatibility, the code modules each contain a **revision number.** The revision numbers can be found by using the `what` command. The following is an example of how this call can be used:

```
$ what /usr/lib/libsb1.a
/usr/lib/libsb1.a:
     41.2     1/17/86 libsb1.a
$ what /usr/lib/libsb2.a
/usr/lib/libsb2.a:
     41.2     1/17/86 libsb2.a
$ what /usr/lib/libddhil.a
/usr/lib/libddhil.a:
     41.2     1/17/86 libhil.a
$
```

Figure INTRO-COMP-2.

The following table indicates compatibility between the high level Starbase code and the Starbase driver code for the Series 300 computers.

In the following table the Starbase and Driver Revision numbers correspond to the following HP-UX Release numbers:

Note An "*x*" in the following tables indicates all versions of that number are applicable. Example: 50.*x* indicates 50.1, 50.2, etc.

Table INTRO-COMP-2.

Revision Numbers	HP-UX Release
28.1	5.0
39.1	5.1
50.*x*	5.18
65.*x*	5.2
65.1.1.*x*	5.3
65.1.3.1	5.5
110.1	6.0
150.1.2.1	6.2
250.1.2.1	6.5
300.1.2.1	7.0

Table INTRO-COMP–3. Series 300 Driver and Revision Numbers

Device Driver Section	Driver Library	Released Starbase Revisions
HP-GL Device Driver	`libddhpgl.a`	28.1, 39.1, 50.*x*, 65.*x*, 65.1.3.1, 110.1, 150.1.2.1, 250.1.2.1, 300.1.2.1
CADplt Device Driver	`libddCADplt.a`	110.1, 150.1.2.1, 250.1.2.1, 300.1.2.1
CADplt2 Device Driver	`libddCADplt.a`	300.1.2.1
CGM Device Driver	`libddhpcgm.a`	150.1.2.1, 250.1.2.1, 300.1.2.1
HP-HIL Device Driver	`libddhil.a`	28.1, 39.1, 50.*x*, 65.*x*, 65.1.1.*x*, 65.1.3.1, 110.1, 150.1.2.1, 250.1.2.1, 300.1.2.1
HP Keyboard Device Driver	`libddkbd.a`	28.1, 39.1, 50.*x*, 65.*x*, 65.1.1.*x*, 65.1.3.1, 110.1, 150.1.2.1, 250.1.2.1, 300.1.2.1
HP Locator Keyboard Device Driver	`libddlkbd.a`	110.1, 150.1.2.1, 250.1.2.1, 300.1.2.1
HP Starbase Memory Driver	`libddSMDpix.a` `libddSMDplane.a`	65.*x*, 65.1.1.*x*, 65.1.3.1, 110.1, 150.1.2.1, 250.1.2.1, 300.1.2.1
HP Starbase-on-X11 Device Driver	`libddsox11.a`	150.1.2.4, 250.1.2.1, 300.1.2.1
HP Terminal Device Driver	`libddhpterm.a`	39.1, 50.*x*, 65.*x*, 65.1.1.*x*, 65.1.3.1, 110.1, 150.1.2.1, 250.1.2.1, 300.1.2.1
HP Windows/9000 Device Driver	`libddbyte.a`	65.*x*, 65.1.1.*x*, 65.1.3.1, 110.1, 150.1.2.1, 250.1.2.1, 300.1.2.1

Table INTRO-COMP-3. Series 300 Driver and Revision Numbers Continued

Device Driver Section	Driver Library	Released Starbase Revisions
HP 300H Device Driver	`libdd300h.a`	28.1, 39.1, 65.*x*, 65.1.1.*x*, 65.1.3.1, 110.1, 150.1.2.1, 250.1.2.1, 300.1.2.1
HP 300L Device Driver	`libdd300l.a`	28.1, 39.1, 65.1.1.*x*, 65.1.3.1, 110.1, 150.1.2.1, 250.1.2.1, 300.1.2.1
HP 98700 Device Driver	`libdd98700.a`	28.1, 39.1, 65.*x*, 65.1.3.1, 110.1, 150.1.2.1, 250.1.2.1, 300.1.2.1
HP 98710 Device Driver	`libdd98710.a`	28.1, 39.1, 65.*x*, 65.1.3.1, 110.1, 150.1.2.1, 250.1.2.1, 300.1.2.1
HP 98550 Device Driver	`libdd98550.a`	65.1.3.1, 110.1, 150.1.2.1, 250.1.2.1, 300.1.2.1
HP 98556 Device Driver	`libdd98556.a`	110.1, 150.1.2.1, 250.1.2.1, 300.1.2.1
HP 98720 Device Driver	`libdd98720.a`	50.*x*, 65.*x*, 65.1.1.*x*, 65.1.3.1, 110.1, 150.1.2.1, 250.1.2.1, 300.1.2.1
HP 98720w Device Driver	`libdd98720w.a`	50.*x*, 65.*x*, 65.1.3.1, 110.1, 150.1.2.1, 250.1.2.1, 300.1.2.1
HP 98721 Device Driver	`libdd98721.a`	50.*x*, 65.*x*, 65.1.3.1, 110.1, 150.1.2.1, 250.1.2.1, 300.1.2.1
X10/Xn Windows Device Driver	`libddXn.a`	65.1.3.1, 110.1, 150.1.2.1, 250.1.2.1, 300.1.2.1
HP 98730 Device Driver	`libdd98730.a`	150.1.2.1, 250.1.2.1, 300.1.2.1
HP 98731 Device Driver	`libdd98731.a`	150.1.2.1, 250.1.2.1, 300.1.2.1

The following table indicates compatibility between the high level Starbase code and the Starbase driver code for the Series 800 computers.

In the next table (Series 800 Drivers and Revision Numbers) the Starbase and driver revision numbers correspond to the following HP-UX release numbers:

Note An "*x*" in the following tables indicates all versions of that number are applicable. Example: 48.*x* indicates 48.1, 48.2, etc.

Table INTRO-COMP–4.

Revision Numbers	HP-UX Release
48.*x*	1.0
80.*x*	1.1
83.*x*	1.2
120.*x*	2.0, 2.1
200.1.10.1	3.0
270.1.2.1	3.1
300.1.2.1	7.0

Table INTRO-COMP-5. Series 800 Driver and Revision Numbers

Device Driver Section	Driver Library	Released Starbase Revisions
HP CADplt Device Driver	`libddCADplt.a`	120.*x*, 200.1.10.1, 270.1.2.1, 300.1.2.1
HP CADplt2 Device Driver	`libddCADplt.a`	300.1.2.1
HP-GL Device Driver	`libddhpgl.a`	48.*x*, 80.*x*, 83.*x*, 120.*x*, 200.1.10.1, 270.1.2.1, 300.1.2.1
HP-HIL Device Driver	`libddhil.a`	80.*x*, 83.*x*, 120.*x*, 200.1.10.1, 270.1.2., 300.1.2.1
HP Keyboard Device Driver	`libddkbd.a`	48.*x*, 80.*x*, 83.*x*, 120.*x*, 200.1.10.1, 270.1.2.1, 300.1.2.1
HP Locator Keyboard Device Driver	`libddlkbd.a`	120.*x*, 200.1.101, 270.1.2.1, 300.1.2.1
HP Starbase Memory Driver	`libddsmdpix.a`	80.*x*, 83.*x*, 120.*x*, 200.1.10.1, 270.1.2.1, 300.1.2.1
HP Starbase Memory Driver	`libddsmdplane.a`	120.*x*, 200.1.10.1, 270.1.2.1, 300.1.2.1
HP Starbase-on-X11 Device Driver	`libddsox11.a`	200.1.10.1, 270.1.2.1, 300.1.2.1
HP Terminal Device Driver	`libddhpterm.a`	48*x*, 80.*x*, 83.*x*, 120.*x*, 200.1.10.1, 270.1.2.1, 300.1.2.1
HP Terminal Device Driver	`libdd262x.a`	48.*x*, 80.*x*, 83.*x*, 120.*x*, 200.1.10.1, 270.1.2.1, 300.1.2.1
HP X10/Xn Windows Device Driver	`libddXn.a`	80.*x*, 83.*x*, 120.*x*, 200.1.10.1, 270.1.2.1, 300.1.2.1

Table INTRO-COMP–5. Series 800 Driver and Revision Numbers Continued

Device Driver Section	Driver Library	Released Starbase Revisions
HP 98550 Device Driver	`libdd98550.a`	83.*x*, 120.*x*, 200.1.10.1, 270.1.2.1, 300.1.2.1
HP 98556 Device Driver	`libdd98556.a`	120.*x*, 200.1.10.1, 270.1.2.1, 300.1.2.1
HP 98720 Device Driver	`libdd98720.a`	80.*x*, 83.*x*, 120.*x*, 200.1.10.1, 270.1.2.1, 300.1.2.1
HP 98720w Device Driver	`libdd98720w.a`	80.*x*, 83.*x*, 120.*x*, 200.1.10.1, 270.1.2.1, 300.1.2.1
HP 98721 Device Driver	`libdd98721.a`	80.*x*, 83.*x*, 120.*x*, 200.1.10.1, 270.1.2.1, 300.1.2.1
HP 98730 Device Driver	`libdd98730.a`	200.1.10.1, 270.1.2.1, 300.1.2.1
HP 98731 Device Driver	`libdd98731.a`	200.1.10.1, 270.1.2.1, 300.1.2.1
HP-CGM Device Driver	`libddhpcgm.a`	200.1.10.1, 270.1.2.1, 300.1.2.1

Note An "*x*" in the following tables indicates all versions of that number are applicable. Example: 50.*x* indicates 50.1, 50.2, etc.

Table INTRO-COMP-6. Series 300 Formatter and Revision Numbers

Formatter Section	Formatter Library	Starbase Revision	Formatter Revision
HP Printer Control Language (PCL) Formatter	`libfmtpcl.a`	65.*x*, 65.1.3.1, 110.1, 150.1.2.1, 300.1.2.1	65.*x*, 65.1.3.1, 110.1, 150.1.2.1, 300.1.2.
HP Printer Control Language (PCL) with imaging extensions	`libfmtpcl.a`	300.1.2.1	300.1.2.1
Versatec C2500 Formatter	`libfmtvers.a`	50.*x*[1]	2.*x*

[1] The Versatec Formatter works with all versions of Starbase after 50.*x*.

Note

The Versatec C2500 Formatter is not part of Starbase and is obtained by ordering the HP 98053A product. The HP 98053A product requires a HP 98622A GPIO card which must be ordered separately.

The HP 9000 Model 310 should be used to drive *only* the C2552, or C2562 200 dpi Versatec plotters. The HP 9000 Models 320, 330, 350, 360, or 370 may be used to drive any of the Versatec C2552, C2558, C2562, or C2568 plotters.

Note An "*x*" in the following tables indicates all versions of that number are applicable. Example: 80.*x* indicates 80.1, 80.2, etc.

Table INTRO-COMP-7. Series 800 Formatter and Revision Numbers

Formatter Section	Formatter Library	Starbase Revision	Formatter Revision
HP Printer Control Language (PCL) Formatter	`libfmtpcl.a`	80.*x*, 83.*x*, 120.*x*, 200.1.10.1, 300.1.2.1	80.*x*, 83*x*, 120*x*, 200.1.10.1, 300.1.2.1
HP Printer Control Language (PCL) with imaging extensions	`libfmtpcl.a`	300.1.2.1	300.1.2.1

Note No direct connection of the HP 9000 Series 800 to Versatec plotters is supported by Hewlett-Packard. Versatec hardcopy for Series 800 is supported indirectly by spooling to an HP 9000 Series 300. PCL color hardcopy is supported by the `pcltrans` command and the `libfontpcl.a` formatter.

Table INTRO-COMP–8. Starbase Device Drivers Supported on Series 300

Drivers	310	318	319C+	319SRX	320	330	340	350	360	370
HP-CGM	No	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes
HP-HIL	No	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes
HP-GL CADplt	No	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes
HP-GL/2 CADplt2	No	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes
HP Keyboard HP Locator Keyboard	No	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes
Memory	No	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes
SOX11	No	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes
HP Terminal	No	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes
HP Windows/9000	No	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes
HP 300H HP 300L	No	Note[1]	No	No	Yes	Yes	Note[1]	Yes	Yes	Yes
HP 9836A	No	No	No	No	Yes	Yes	No	Yes	Yes	Yes
HP 98550 HP 98556	No	No	Note[2]	No	No	Yes	Yes	Yes	Yes	Yes
HP 98700 HP 98710	No	No	No	No	Yes	Yes	No	Yes	Yes	Yes
HP 98720 HP 98720w HP 98721	No	No	No	Yes	Yes	Yes	Yes	Yes	Yes	Yes
HP 98730 HP 98731	No	No	No	No	No	No	No	Yes	Yes	Yes
X10/Xn Windows	No	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes

1 HP 300H monochrome only.

2 HP 98549 only (HP 98556 is not supported.)

Note The 7.0 release of Starbase does *not* support the Series 310 CPU.

Table INTRO-COMP-9. Starbase Device Drivers Supported on Series 800

Drivers	815	825	835	840	850/855
HP-CGM	Yes	Yes	Yes	Yes	Yes
HP-HIL	Yes	Yes	Yes	Yes	Yes
HP-GL CADplt	Yes	Yes	Yes	Yes	Yes
HP-GL/2 CADplt2	Yes	Yes	Yes	Yes	Yes
HP Keyboard HP Locator Keyboard	Yes	Yes	Yes	Yes	Yes
Memory	Yes	Yes	Yes	Yes	Yes
SOX11	No	Yes	Yes	Yes	Yes
HP Terminal	Yes	Yes	Yes	Yes	Yes
HP 98550 HP 98556	Yes	Yes	Yes	No	No
HP 98720 HP 98720w HP 98721	Yes	Yes	Yes	Yes	No
HP 98730 HP 98731	Yes	Yes	Yes	No	No
X10/Xn Windows	No	Yes	Yes	Yes	Yes

Graphics Libraries Supported Within Windows

The following table shows which graphics libraries run in the different window systems that are supported on the the HP 9000 Series 300 and Series 800 workstations.

Table INTRO-COMP-10. Graphics Libraries Supported in the Different Window Systems

Window Systems	Starbase and Starbase Display List	AGP/ DGL	HP-GKS	Xlib Graphics
Raw Mode	Yes	Yes	Yes	No
HP Windows/9000 (Series 300 only)	Yes	Yes	Yes	No
X10	Yes, via the Xn driver	No	No	Yes
X11 (revision A.00)	Yes, via the SOX11 driver	No	Yes, via the SOX11 driver	Yes
X11	Yes, via the SOX11 driver or the Starbase Display Drivers	No	Yes, via the SOX11 driver or the Starbase Display Drivers	Yes

Graphic Escape Functions (gescapes)

The Starbase graphics library provides a very broad range of capabilities; however, there are cases where a particular device has capabilities not directly supported by Starbase. In this case, a **graphics escape sequence procedure** (`gescape`) may be provided to permit access to these capabilities. The `gescape` procedures should be used with care. Each driver section contains a list of supported `gescape` procedures plus complete details on those procedures unique to that driver. Details of the non-unique procedures are provided in the appendix.

Note

A `gescape` capability that is currently supported on a device may not be supported on future devices in that same class (e.g., high-resolution displays). Furthermore, for a particular device, `gescape` operations may change depending on other system requirements. For example, offscreen display memory is used differently with and without windows; therefore, direct access to this memory using `gescapes` may return different data.

Contents

CADPLT

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` library. This driver provides hardware support for certain areas of functionality for Starbase graphics. 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 CADPLT-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 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` protocal with `dc1` and `dc3` signals
    - ";" command terminator
    - ⟨*newline*⟩ response terminator
2. device interface, `termio`(4)
    - 8-bit character size
    - `XON/XOFF` protocal
    - no parity

- disabled INTR and QUIT signals
- 2400 baud rate if initially 300 [1]
- no postprocessing
- canonical processing
- undefine 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 1000000 < /dev/plts &
```

Note

If the device is in SPOOLED mode, the device interface `termio`(4) will *not* be automatically configured for the user. It is the users 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.

---

[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 the baud rate has been changed from the default value of 300 by the user, then the driver assumes the user has purposely changed it and will not modify it.

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

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 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

This 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 `-lddCADplt` linking option. If you link in `libddCADplt.a`, you must also link in `libdvio.a` as below:

```
cc example.c -lddCADplt -lsb1 -lsb2 -ldvio
```

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 Fortan 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 CADPLT-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. The user should consult his 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, 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 CADPLT-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 user's 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 gopened with `INT_XFORM`.

Table CADPLT-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
35	iso_united	ISO United Kingdom
36	iso_italian	ISO Italian
37	iso_spanish	ISO Spanish
38	iso_portugues	ISO Norway, Version 2
39	iso_norway_v2	ISO Norway, Version 2
60	iso_french_v2	ISO French
99	iso_drafting	Drafting Symbols
100	kanji_v1	Kanji, part 1
101	kanji_v2	Kanji, part 2

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`. The user 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`".

[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 CADPLT-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
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`. The user 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 CADPLT-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	⟨*index*⟩ = 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.

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

If pen number 0 is selected, then a device dependent action will occur and the user should consult his 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 feed out. Devices supporting this functionality are shown below:

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

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 CADPLT-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.

```
await_retrace
backface_contro
background_color
background_color_index
bank_switch
bf_control
bf_fill_color
bf_interior_style
bf_perimeter_color
bf_perimeter_repeat_length
bf_perimeter_type
bf_surface_coefficients
bf_surface_model
block_move
block_read
block_write
clear_control
dbuffer_switch
dcblock_move
dcblock_read
dcblock_write
define_raster_echo
define_trimming_curve
depth_cue
depth_cue_color
depth_cue_range
display_enable
double_buffer
drawing_mode
fill_dither
hidden_surface
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
pattern_define
shade_mode
shade_range
surface_model
surface_coefficients
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.
`vertex_format`	the ⟨*use*⟩ parameter must be zero, any extra coordinates supplied will be ignored.

Parameters for gescape

The following gescape functions are common to two or more drivers and are discussed in the appendix of this manual:

`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.

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

Contents

CADPLT2

The CADplt2 Device Driver

Device Description

The driver library `libddCADplt.a` 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. 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 CADPLT2-1. 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 widelines	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 then 110, otherwise two 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` protocal 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` protocal
    - no parity

- disabled INTR and QUIT signals
- 2400 baud rate if initially 300 [1]
- no postprocessing
- canonical processing
- undefine 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 1000000 < /dev/plts &
```

Note

If the device is in SPOOLED mode, the device interface `termio(4)` will *not* be automatically configured for the user. It is the users 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.

---

[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 the baud rate has been changed from the default value of 300 by the user, then the driver assumes the user has purposely changed it and will not modify it.

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

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
```

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

This 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 `-lddCADplt` linking option. If you link in `libddCADplt.a`, you must also link in `libdvio.a` as below:

```
cc example.c -lddCADplt -lsb1 -lsb2 -ldvio
```

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

For C programs:

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

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

For Fortan 77 programs:

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

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

For 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.

---

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

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.

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 CADPLT2-2. 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 CADPLT2-3. 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 the user's responsibility to insure that appropriate P1 and P2 values are currently established.

- RESET_DEVICE mode

  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

[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.

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 gopened with `INT_XFORM`.

Table CADPLT2-4. 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	hpturkish8	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 CADPLT2–5. 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

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 CADPLT2–6. HP-GL/2 Typefaces

gescape Argument	Typeface	gescape 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

Table CADPLT2-6. HP-GL/2 Typefaces
Continued

gescape Argument	Typeface	gescape Argument	Typeface
27	itc_korinna	70	(available)
28	naskh	71	itc_american_typewriter
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. The user can *not* turn off hardware support

of polygons.[4] Consult your device's hardware manual for the actual number of vertices supported.

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 CADPLT2-7. 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`	⟨*index*⟩ = 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

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 Starbse font indices in the following table; however, for a more

[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.

extensive font typeface selection, use the `gescape` provided to access all the HP-GL/2 typefaces.

Table CADPLT2-8. 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 and 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 CADPLT2-9. Supported Combinations of text_path and text_line_path

	text_line_path			
text_path	`path_right`	`path_left`	`path_up`	`path_down`
`path_ritht`	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 CADPLT2-10. 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.

```
await_retrace                  depth_cue_range
backface_contro                display_enable
background_color               double_buffer
background_color_index         drawing_mode
bank_switch                    fill_dither
bf_control                     hidden_surface
bf_fill_color                  intblock_move
bf_interior_style              intblock_read
bf_perimeter_color             intblock_write
bf_perimeter_repeat_length     interior_style (INT_OUTLINE)
bf_perimeter_type              interior_style (INT_POINT)
bf_surface_coefficients        intline_width
bf_surface_model               light_ambient
block_move                     light_attenuation
block_read                     light_model
block_write                    light_source
clear_control                  light_switch
dbuffer_switch                 pattern_define
dcblock_move                   shade_mode
```

`dcblock_read`	`shade_range`
`dcblock_write`	`surface_model`
`define_raster_echo`	`surface_coefficients`
`define_trimming_curve`	`viewpoint`
`depth_cue`	`write_enable`
`depth_cue_color`	`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.

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

Parameters for gescape

The following gescape functions are common to two or more drivers and are discussed in the appendix of this manual:

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

The following gescape functions are unique to this driver and a detailed discussion of each concludes this chapter.

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.

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.[6]

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

```
      integer*4 arg1i(64), arg2i(64)
C
C Enable cutter
C
      arg1i(1) = 1
      call gescape(fildes,HPGL2_CUTTER_CONTROL,arg1i,arg2i)
```

[6] 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 corresponding to the desired typeface.[7]

The arg2 parameter is ignored.

C Syntax Examples

```
#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 Examples

```
      integer*4 arg1i(64), arg2i(64)
C
C Select presentations typeface
C
      arg1i(1) = 11
      call gescape(fildes,HPGL2_FONT_TYPEFACE,arg1i,arg2i)
```

[7] See the table "Typefaces" earlier in this chapter for recognized typefaces for HP-GL/2.

Pascal Syntax Examples

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

var
    arg1, arg2 : gescape_arg;

begin

    { Select presentations typface }

    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.[8]

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.f[0] = 6.0;
arg2.i[0] = 3;
gescape(fildes,HPGL2_LOGICAL_PEN_WIDTH,&arg1,&arg2);
```

FORTRAN77 Syntax Example

```
      real arg1f(64)
      interger*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)
```

---

[8] Whether arg1 represents a physical pen number or a color map entry depends on the device. Electrostatic plotters have no physical pens.

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;
    arg2.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 plotter 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 contain 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 on 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 takes a single integer parameter in arg1 between 0 and 100 which 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 any 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 preety 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);
```

Contents

HPCGM

HP CGM Device Driver

Device Description

This device driver 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.

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 standards, 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.

Linking the Driver

The `hpcgm` driver is located in the `/usr/lib` directory with the file name `libddhpcgm.a`. This driver may be linked to a program by using the absolute path `/usr/lib/libddhpcgm.a` or an appropriate relative path name, or by using the `-l` option `-lddhpcgm`. For example, to compile and link a program for use with this driver, use:

```
cc example.c -lddhpcgm -lsb1 -lsb2 -o example
fc example.f -lddhpcgm -lsb1 -lsb2 -o example
pc example.p -lddhpcgm -lsb1 -lsb2 -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 HPCGM-1. Default Color Tabrle

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
18to255	shaded colors			

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

Table HPCGM-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.

```
await_retrace
backface_control
bank_switch
bf_control
bf_fill_color
bf_interior_style
bf_perimeter_color
bf_perimeter_repeat_length
bf_perimeter_type
bf_surface_coefficients
bf_surface_model
block_move
block_read
block_write
clear_control
dbuffer_switch
dcblock_move
dcblock_read
dcblock_write
dcecho_type
dcecho_update
define_raster_echo
define_trimming_curve
depth_cue
depth_cue_color
depth_cue_range
display_enable
double_buffer
drawing_mode
echo_type
echo_update
fill_dither
hidden_surface
inquire_hit
inqiure_pick_depth
inquire_pick_window
interior_style (INT_OUTLINE)
interior_style (INT_POINT)
light_ambient
light_attenuation
light_model
light_source
light_switch
pattern_define
set_hit_mode
set_pick_depth
set_pick_window
shade_range
surface_coefficients
surface_model
track
track_off
viewpoint
write_enable
zbuffer_switch
```

Conditionally Supported

The following Starbase commands are supported under the listed conditions:

`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.
`clear_view_surface`	This causes completion of a previous picture and begins a new picture in metafile.

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`.

The following sections give details on each of the `gescape` functions.

CGMESC_APPL_DATA

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

This `gescape` generates a 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

```
gescape_arg  arg2;
   :
gescape(fildes, CGMESC_APPL_DATA,
    "10   APPLICATION move/draw", &arg2);
```

FORTRAN Syntax

```
 character arg2(255)
    :
 call gescape(fildes, CGMESC_APPL_DATA,
+  '10   APPLICATION move/draw'//char(0), arg2)
```

Pascal Syntax

```
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 the CGM encoding style.

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

```
gescape_arg  arg2;
   :
gescape(fildes, CGMESC_ENCODING, "T", &arg2);
```

FORTRAN Syntax

```
character arg2(255)
   :
   :call gescape(fildes, CGMESC_ENCODING, 'T'//char(0), arg2)
```

Pascal Syntax

```
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

```
gescape_arg  arg2;
   :
gescape(fildes, CGMESC_ESCAPE_ELT, "-302  1.0 0.0 0.0 0.22", &arg2);
```

FORTRAN Syntax

```
 character arg2(255)
    :
 call gescape(fildes, CGMESC_ESCAPE_ELT,
+  '-302  1.0 0.0 0.0 0.22'//char(0), arg2)
```

Pascal Syntax

```
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

```
gescape_arg  arg2;
   :
gescape(fildes, CGMESC_FONT_IX, "12", &arg2);
```

FORTRAN Syntax

```
character arg2(255)
   :
call gescape(fildes, CGMESC_FONT_IX, '12'//char(0), arg2)
```

Pascal Syntax

```
var arg1, arg2: gescape_arg;
   :
arg1.c := '12'#0;
gescape(fildes, CGMESC_FONT_IX, arg1, arg2);
```

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.

The arg2 parameter is ignored.

C Syntax

```
gescape_arg  arg2;
   :
gescape(fildes, CGMESC_MESSAGE, "Next is the move/draw polygon", &arg2);
```

FORTRAN Syntax

```
 character arg2(255)
    :
 call gescape(fildes, CGMESC_MESSAGE,
+  'Next is the move/draw polygon'//char(0), arg2)
```

Pascal Syntax

```
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

```
gescape_arg  arg2;
   :
gescape(fildes, CGMESC_MET_NAME, "HP-CGM metafile name", &arg2);
```

FORTRAN Syntax

```
 character arg2(255)
    :
 call gescape(fildes, CGMESC_MET_NAME,
+  'HP-CGM metafile name'//char(0), arg2)
```

Pascal Syntax

```
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

```
gescape_arg  arg2;
   :
gescape(fildes, CGMESC_PIC_NAME, "Picture name", &arg2);
```

FORTRAN Syntax

```
 character arg2(255)
    :
 call gescape(fildes, CGMESC_PIC_NAME,
+  'Picture name'//char(0), arg2)
```

Pascal Syntax

```
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

```
gescape_arg  arg1,arg2;
   :
gescape(fildes, CGMESC_TOP_MODE, &arg1, &arg2);
```

FORTRAN Syntax

```
character arg1(255), arg2(255)
   :
call gescape(fildes, CGMESC_TOP_MODE, arg1, arg2)
```

Pascal Syntax

```
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

```
gescape_arg  arg2;
   :
gescape(fildes, CGMESC_VDC_PREC, "L", &arg2);
```

FORTRAN Syntax

```
character arg2(255)
   :
call gescape(fildes, CGMESC_VDC_PREC, 'L'//char(0), arg2)
```

Pascal Syntax

```
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)

Contents

HP-GL

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.

Series 300

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/plotter 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/plotter c 1 0x⟨sc⟩⟨ad⟩04
```

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/plotter 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/plotter c 1 0x00⟨lu⟩⟨ad⟩
```

Linking the Driver

The 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`. If you use (link) `libddhpgl.a`, you must also include (link) the `libdvio.a` file. For example, to compile and link a program for use with this driver, use:

```
cc example.c -lddhpgl -ldvio -lsb1 -lsb2 -o example
fc example.f -lddhpgl -ldvio -lsb1 -lsb2 -o example
pc example.p -lddhpgl -ldvio -lsb1 -lsb2 -o example
```

depending upon the language being used.

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/plotter", 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 HP-GL-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 HP-GL–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 HP-GL-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.

```
await_retrace
backface_control
background_color
background_color_index
bank_switch
bf_control
bf_fill_color
bf_interior_style
bf_perimeter_color
bf_perimeter_repeat_length
bf_perimeter_type
bf_surface_coefficients
bf_surface_model
block_move
block_read
block_write
clear_control
dbuffer_switch
dcblock_move
dcblock_read
dcblock_write
define_raster_echo
define_trimming_curve
depth_cue
depth_cue_color
depth_cue_range
display_enable
double_buffer
drawing_mode
fill_dither
hidden_surface
intline_width
intblock_move
intblock_read
intblock_write
interior_style (INT_OUTLINE)
interior_style (INT_POINT)
light_ambient
light_attenuation
light_model
light_source
light_switch
line_endpoint
pattern_define
shade_mode
shade_range
surface_model
surface_coefficients
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).

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

`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.

Parameters for gescape

The following gescape functions are common to two or more drivers and discussed in the appendix of this manual.

- `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.

Contents

HP-HIL

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⟨*ad*⟩ where ⟨*ad*⟩ is the device's two digit address (position on the HP-HIL loop from the computer interface card).

```
mknod /dev/hilx c 24 0x0000⟨ad⟩
```

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*⟩⟨*ad*⟩ where ⟨*lu*⟩ is the two-digit hardware logical unit and ⟨*ad*⟩ is the device's two-digit address (position from the computer interface card). Note that the `0x` causes the number to be interpreted hexadecimally.

`mknod /dev/hilx c 24 0x00`⟨*lu*⟩⟨*ad*⟩

or

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

Linking the Driver

The 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`. For example, to compile and link a program for use with this driver, use:

```
cc example.c -lddhil -lsb1 -lsb2 -o example
fc example.f -lddhil -lsb1 -lsb2 -o example
pc example.p -lddhil -lsb1 -lsb2 -o example
```

depending upon the language being used. To use this device driver in an X11 window, the libraries `/usr/lib/libXwindow.a`, `/usr/lib/libXhp11.a` and `/usr/lib/libX11.a` must also be linked to the program. For example, to compile and link a program for use in an X11 window, use:

```
cc example.c -lddhil -lXwindow -lsb1 -lsb2 -lXhp11 -lX11 -o example
fc example.f -lddhil -lXwindow -lsb1 -lsb2 -lXhp11 -lX11 -o example
pc example.p -lddhil -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 Programming with X11* 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:

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

Mode — This parameter is ignored.

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 devices

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.

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 [f1] thru [f8] plus the four unmarked keys in the upper-right corner of the keyboard representing keys 1 thru 12 respectively.

Table HP-HIL-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
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 following gescape functions are common to multiple device drivers. Detailed information about these 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 and are presented only in this section.

- 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.

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);
```

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)
```

Pascal Syntax Example

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

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);
```

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);
```

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

```
/* 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)
```

Pascal 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

```
/* 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)
```

Pascal Syntax Example

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

```
/* 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);
```

Pascal Syntax Example

```
{gescape_arg is defined in starbase.p1.h}
var
   arg1, agr2: gescape_arg;
begin
   gescape (fildes, REPORT_PROXIMITY, arg1, arg2);
```

SET_ACCELERATION

The ⟨*op*⟩ 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

```
/* 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)
```

Pascal 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);
```

Contents

KBD

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

The 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 `-lddkbd`. For example, to compile and link a program for use with this driver, use:

```
cc example.c -lddkbd -lsb1 -lsb2 -o example
fc example.f -lddkbd -lsb1 -1sb2 -o example
pc example.p -lddkbd -lsb1 -lsb2 -o example
```

depending upon the language being used.

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.

Contents

LKBD

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

The 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 -lddlkbd. The driver also requires the curses(3) library to be linked. For example, to compile and link a program for use with this driver, use:

```
cc example.c -lddlkbd -lsb1 -lsb2 -lcurses -o example
```

or

```
fc example.f -lddlkbd -lsb1 -lsb2 -lcurses -o example
pc example.p -lddlkbd -lsb1 -lsb2 -lcurses -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 LKBD-1.

Capability Description	Capability ID
enable keypad	smkx
disable keypad	rmkx
up arrow.	kuu1
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 accordingly.

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 unique to this driver and are discussed in this section:

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

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

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

gescape_arg arg1, arg2;
      :
gescape(fildes,ENABLE_ACKNOWLEDGE,&arg1,&arg2);
```

FORTRAN77 Syntax

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

Pascal Syntax

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

var
   arg1,arg2:gescape_arg;
begin
   gescape(fildes,ENABLE_ACKNOWLEDGE,arg1, arg2);
```

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

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

gescape_arg arg1, arg2;
      :
gescape(fildes,DISABLE_ACKNOWLEDGE,&arg1,&arg2);
```

FORTRAN77 Syntax

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

Pascal Syntax

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

var
   arg1,arg2:gescape_arg;
begin
   gescape(fildes,DISABLE_ACKNOWLEDGE,arg1, arg2);
```

Contents

HPTERM

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 `0x00`⟨*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

The 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`. To compile and link a program for use with this driver use:

```
cc example.c -lddhpterm -lsb1 -lsb2 -o example
fc example.f -lddhpterm -lsb1 -lsb2 -o example
pc example.p -lddhpterm -lsb1 -lsb2 -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/ttyxx`.) 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 consists 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 HPTERM-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 HPTERM-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 *HP-UX Reference,* "Procedure `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.

This device driver defines line type 6 to be the same as line type 3.

This device driver defines line type −1 as terminal line type 11 (point plotting).

This device driver defines 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.

```
await_retrace                 intblock_move
backface_control              intblock_read
bank_switch                   intblock_write
bf_control                    interior_style (INT_OUTLINE)
bf_fill_color                 interior_style (INT_POINT)
bf_interior_style             intline_repeat_length
bf_perimeter_color            intline_width
bf_perimeter_repeat_length    intperimeter_repeat_length
bf_perimeter_type             light_ambient
bf_surface_coefficient        light_attenuation
bf_surface_model              light_model
block_move                    light_source
block_read                    light_switch
block_write                   line_endpoint
dbuffer_switch                line_repeat_length
dcblock_move                  pattern_define
dcblock_read                  perimeter_repeat_length
dcblock_write                 shade_mode
define_raster_echo            shade_range
define_trimming_curve         surface_model
depth_cue                     surface_coefficients
depth_cue_color               track
depth_cue_range               track_off
display_enable                viewpoint
double_buffer                 write_enable
hidden_surface                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.

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 will not be 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.

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.

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.

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.

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 HPTERM-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 `READ_COLOR_MAP gescape` is common to two or more devices and discussed in Appendix A.

The following `gescape` functions are unique for this driver and discussed on the next pages.

- `HPTERM_640x400`—Set high-resolution spooled output
- `HPTERM_PRINT_ESC` or `HP26_PRINT_ESC`—Send terminal control (escape) strings

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 Example

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

int fildes;
gescape_arg arg1, arg2;

fildes = gopen("/dev/tty", OUTDEV, "hp2393", INIT | SPOOLED);
gescape(fildes, HPTERM_640x400, &arg1, &arg2);
set_p1_p2(fildes, FRACTIONAL, 0.0, 0.0, 0.0, 1.0, 1.0, 1.0);
```

FORTRAN77 Example

```
integer*4 fildes,arg1(64),arg2(64)

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 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, arg1, arg2);
  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 **arg1** 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 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 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 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.
```

Contents

WIN9000

HP Windows/9000

Device Description

HP Windows/9000 supports two types of windows:

- the Terminal Window Type
- the Graphics Window Type

HP Windows/9000 is only supported on the Series 300 and Series 500 computers.

This chapter discusses graphics windows (also called Starbase windows), that are supported by the following Starbase display drivers:

`hp300h`	HP 300 High-resolution Device Driver
`hp300l`	HP 300 Medium-resolution Device Driver
`hp98550`	HP 98550 Device Driver
`hp98556`	HP 98556 Device Driver
`hp98700`	HP 98700 Device Driver
`hp98720W`	HP 98720w Device Driver
`hp98730`	HP 98730 Device Driver
`hp98731`	HP 98731 Device Driver

Note Only the HP 98720w Device Driver supports HP Windows/9000. The HP 98720 and HP 98721 drivers do *not* support windows, although they can be used with windows. See the appropriate driver sections for more information.

Note The `hp98730` driver can be used with either normal graphics windows or image graphics windows, a special kind of graphics window. See HP *Windows/9000 Documentation* for more information on image graphics windows.

Note The hp98731 driver can only be used with image windows.

To use graphics windows with any of the above drivers, an additional module called `libwindow.a` must be linked into your program. Furthermore, to support retained rasters for obscured windows, the module `libddbyte.a` or `libddbit.a` must also be linked into your program. Linking procedures are discussed in the "Linking the Driver" segment of this section.

Using graphics windows has performance implications as well as implications on how each process should share the display and display resources (for example, the color map) so that other processes aren't adversely affected. Both of these topics are discussed later in this section.

Device coordinate location (0, 0) is the upper-left corner of the device, regardless of whether the device is a graphics window type (hereafter referred to as `window`) or the normal (hereafter referred to as `raw`) device. The axes are the same whether you are using a `window` device or the `raw` device, with the exception that the lower-right corner is dependent on the window's width and height when created. That is, the location of the lower-right corner in a window will be (width−1, height−1).

Unlike a single graphics process running to a `raw` device, a window graphics process must be a "good citizen" in the graphics world. Since the Window Manager and Term0 Server are built on the Starbase Graphics Library, they will cooperate with other graphics processes that are also "good citizens."

A "bad citizen" is a process that changes global resources that are not process dependent. Examples of these resources are:

- color map values
- planes displayed
- planes blinking

For example, if you set up Windows/9000 to use one set of color map entries, and a second process changes those color map entries to a different set of colors, the new colors may make Windows/9000 difficult to use (for example, black characters on a black background).

Good processes generally have the following characteristics:

- Use gescapes sparingly and in a way that is considerate to other processes.
- Do not change the values of the color map.
- Do not blink planes.
- Do not turn Starbase clipping off, since this will allow access outside the window boundaries, and could, in rare circumstances, cause the window manager to abort. The exception here is when the user is *positive* the `vdc_extent`/device-bounds will *never be exceeded.*

You may want to use double buffering in a graphics program. If so, first ensure that the `0x10` bit in the `WMCONFIG` environment variable is set. You should verify this using `wminquire`(3W) (see the HP *Windows/9000 Documentation*). When a `gopen` is performed by the program, the mode argument should not include `INIT` or the color map will reset. By convention, do a `dbuffer_switch` only while the output window is the selected window.

If you are not going to use double buffering in your program, but double buffering in `WMCONFIG` is set, modify the color indexes the program uses so that the visible color for the graphics output of the program will not change when the display enabled planes are modified by a Starbase program using double buffering. This will prevent your graphics output from blinking when other programs output double buffered programs. The mode used in `gopen` should not include `INIT` in the argument. (See the HP *Windows/9000 User's Manual* for more information.)

Architecture

The Window Manager accesses the display hardware in creating and moving windows. The following diagram shows how Starbase operaties in parallel with the HP Windows/9000 window manager. This architecture permits Starbase to render directly to the display through the display drivers without interacting with the Window Manager, similar to the X11 architecture.

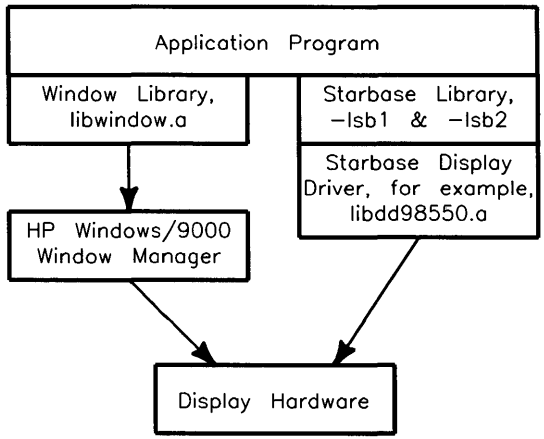


Figure WIN9000-1. Access Display Hardware

Setting Up the Device

Special Device Files

Consult the HP *Windows/9000 Reference* and the HP *Windows/9000 Programmer's Manual* for details on how special files are created for graphics windows.

See the chapter for the device driver you wish to use for information on setting up the device.

Linking the Driver

These device drivers are flexible and can be linked for three different levels of capabilities.

Table WIN9000-1. Linking Requirements

Capability	Libraries Required

Application to raw device only	Device Drivers
	`libsb1`
	`libsb2`

Application to `raw` device or window that refreshes its output occasionally (vector list) and hence does not care if information in window gets lost when window is partly obscured.	Device Drivers
	Window Library
	`libsb1`
	`libsb2`

Application to `raw` device or window that does not refresh its output and hence wants any information in obscured parts of window to be remembered.	Device Drivers
	Byte Driver or
	Bit Driver
	Window Library
	`libsb1`
	`libsb2`

Device drivers are any combination of the Starbase device drivers.

- `libsb1` means `/usr/lib/libsb1.a`
- `libsb2` means `/usr/lib/libsb2.a`
- Byte Driver means `/usr/lib/libddbyte.a`
- Bit Driver means `/usr/lib/libddbit.a`
- Window Library means `/usr/lib/libwindow.a`

The link order is important and should be done as follows:

1. Application object file(s)—required
2. Device driver(s)—at least one required
3. Byte driver and/or bit driver—optional
4. Window library—optional
5. `libsb1`—required
6. `libsb2`—required

Important Use the bit driver instead of the byte driver when using windows on a monochrome display where memory is constrained.

Both the bit driver and the byte driver provide retained raster support for graphics windows. However, the byte driver allocates one byte per pixel while the bit driver allocates one bit per pixel. Thus, the byte driver can use up to eight bits of memory to contain the color information for each pixel. On monochrome displays, only one bit of each byte is used.

Because the bit driver uses a bit per pixel format, eight times less memory is used when monochrome images are stored using this driver. Only monochrome images are stored using the bit driver.

An example of the above note is:

The byte driver will allocate 199,680 bytes of memory to support a raster that is 512×390 pixels on a monochrome display.

The bit driver will only allocate 24,960 bytes to support the retained raster for the same display.

Device 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, or by creating a new graphics window. (For the example following this list, the window device file name `/dev/screen/window1` will be used.)

Window device files are located in the Windows/9000's directory (normally `/dev/screen` or the value of `$WMDIR`, an environment variable), while other Starbase device files, sometimes called `raw` device files, are normally located in the `/dev` directory.

Kind — Specifies the I/O characteristics of the device. This parameter can be one of the following for this device:

- `INDEV`—Input only
- `OUTDEV`—Output only
- `OUTINDEV`—Input and Output

Driver — The character representation of the driver type. This parameter is the same for both `raw` devices and `window` devices; that is, the driver type for a window is the same as the driver type of the `raw` device that the window is on.

Mode — The mode control word that consists of several flag bits *or*ed together. This parameter is the same for either a `raw` or `window` device.

- `0`—same
- `INIT`—Differences for windows are:
  1. Only window area cleared.
  2. If the color map is not its default values because some application changed it, then this resets the color map and keeps a color map dependent application from producing what is intended.
  3. Since Windows/9000 is already running, the display is already enabled.
  4. Should not be used if `WMCONFIG` has the double buffering color mode bit of `0×10` set.

For example, to open and initialize a window for output and input with the directory "/dev/screen" set up to be the directory containing the window device files, and Windows/9000 is running on a HP 98720 display, use:

C Syntax

```
fildes = gopen("/dev/screen/window1",OUTINDEV,"hp98720w",INIT);
```

FORTRAN77 Syntax

```
fildes = gopen('/dev/screen/window1'//char(0),OUTINDEV,
   'hp98720w'//char(0),INIT)
```

Pascal Syntax

```
fildes := gopen('/dev/screen/window1',OUTINDEV,'hp98720w',INIT);
```

Defaults do not differ between a `window` device and the `raw` device.

Performance of Starbase Windows

If the graphics window being drawn to is not obscured by any other objects on the screen, there will be almost no performance degradation when compared to drawing to the `raw` device. If the window is obscured, however, each graphics generation must be clipped to the seen and unseen rectangles for that window. The amount of overhead for this clipping varies with the operation being done. The greater the number of rectangles a window is divided into, the more overhead to do output in that window.

When a window is shrunk smaller than its raster's size, it becomes self-obscured and is divided into three rectangles. If this window is then panned so that none of its edges are seen, it is divided into 5 rectangles.

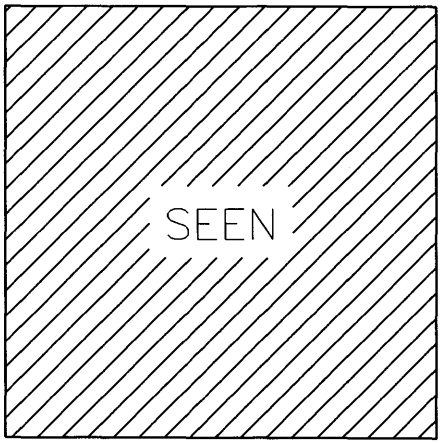


Figure WIN9000–2. Totally Unobscured Window (1 Rectangle)

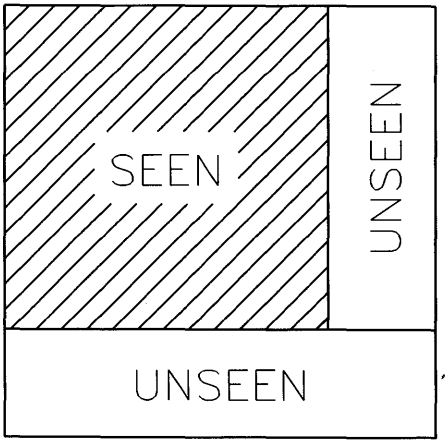


Figure WIN9000–3. Unobscured Window Reduced Size (3 Rectangle)

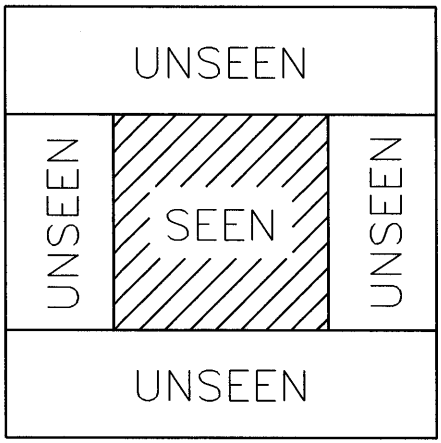


Figure WIN9000–4. Unobscured Window Reduced and Panned

(5 Rectangle)

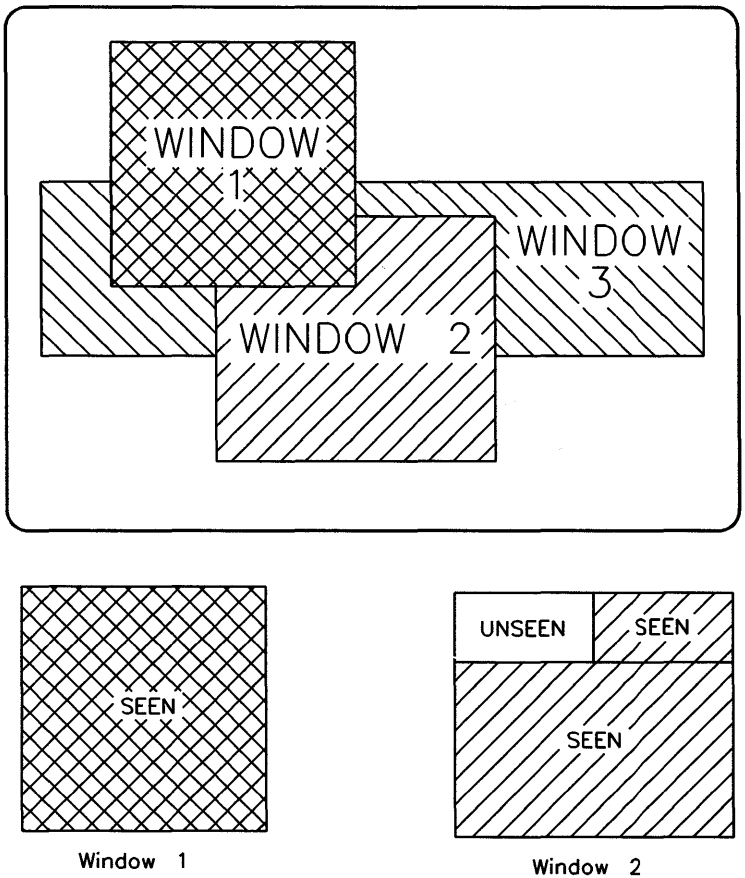


Figure WIN9000–5. Overlapping Windows Divided into Rectangles

Input From Graphics Windows

Graphics windows may be used as input devices, allowing the window system locator device to be shared among applications in different windows. When a graphics window is used for input it has one locator device and two choice devices.

The window locator device reads locations based on the position of Windows/9000 sprite. The window locator device has the same physical limits as the window size, and therefore has the same resolution as the window. The larger a window is, the greater will be the resolution of input values within the window. If the locator is sampled it may return values either inside or outside the window, based on the position of the sprite inside or outside the window. Sampling does not require that the window be selected. Alternatively, the request and event mechanisms wait for a trigger. The trigger for a window device is a button press over the window while the window is selected. The window locator device may be tracked to the same window or any other display. Once tracking from a window is requested, selecting the window will cause the sprite to turn off while over the window and will begin tracking the sprite position to the target display. If the window is not selected, the sprite will remain on over the window and no tracking will occur.

The window choice devices provide two different formats of the same information.

- Choice ordinal 1

  This selection returns button numbers ranging from one to the number of buttons. If the `TRIGGER_ON_RELEASE` `gescape` is enabled, a button release causes choice ordinal 1 to return a negative value of the same magnitude as the button number.

- Choice ordinal 2

  This selection returns a 32 bit wide bit-map. The least significant bit equals button 1 and the most significant bit equals button 32. A one (1) value in a bit indicates that the corresponding button is currently pressed.

Performance of Starbase Input in Windows

Starbase input uses the window library routine `wgetlocator` to get the locator's position. This routine is available to any application that knows it is running in a window. An application doing sampling will get better performance in a window if it uses the `wgetlocator` routine directly, rather than going through the additional Starbase input layer.

Starbase tracking of a window to itself will have lower performance then the window system sprite. An application that tracks a window to itself can get better performance with less cpu usage by using the sprite instead of the track call. The window system call `wsetecho` can be use to change the sprite's appearance inside a window. The sprite can take on the same echo types Starbase provides.

Note

When using the HP 98549A or HP 319C display, do not use a different driver to draw to the windows than is used by the window manager.

For example: Do not use an HP 98550 driver when the window manager uses an HP 300h driver (or vice versa). The drivers manage offscreen frame buffer memory differently and will interfere with each other. The `wminquire`(3W) procedure can be used to programmatically determine what driver the window manager is using.

Fast Alpha and Font Manager Functionality

This device driver supports raster text calls from the fast alpha and font manager libraries. See the *Fast Alpha/Font Manager's Programmer's Manual* for further information.

Parameters for gescape

The following gescape functions are common to other drivers and are discussed in the appendix.

- IGNORE_RELEASE—Trigger when button pressed.
- R_FULL_FRAME_BUFFER—Accesses off-screen area of frame buffer memory.
- R_GET_WINDOW_INFO—Returns window location and status.
- R_LOCK_DEVICE—Locks the specified device.
- R_UNLOCK_DEVICE—Unlocks the specified device.
- SWITCH_SEMAPHORE—Controls semaphore operations.
- TRIGGER_ON_RELEASE—Trigger when button released.

Contents

HP300H

The HP 300H Device Driver

Device Description

This device driver is used with Hewlett-Packard Series 300 high-resolution display systems. See the table in the introduction for supported configurations.

The video board for each of these display systems fits in a SPU system slot. These display systems have a resolution of 1024×768 pixels. The monochrome display system has a single plane of frame buffer. The HP 98545A Color Display System has four planes of frame buffer to provide 16 simultaneous colors. The HP 98547A, HP 98549A, and HP 319C+ Color Display Systems have six planes for 64 colors. A color map provides 8 bits per color (for red,.green and blue), providing a color palette of over 16 million colors.

These display systems are bit-mapped devices with special hardware for:

- Write enabling planes.
- Displaying planes.
- Writing pixels to the frame buffer with a given replacement rule (see `drawing_mode`).
- Blinking planes.
- Moving a block of pixels from one place in the frame buffer to another.

The monochrome and color displays are organized as an array of bytes, with each byte representing a pixel on the display. For the monochrome display, the Least Significant Bit (LSB) of each byte controls the display with 0 for black (pixel off) and 1 for white (pixel on).

For the color displays, the 6 (4 for HP 98545A) LSBs of each byte determine the color, providing color values from 0-63 (0-15 for HP 98545A). These values are used to address the color map. The color map is a RAM table that has 64 (16 for HP 98545A) addressable locations and is 24 bits wide (8 bits each for red, green

and blue). Thus, the pixel value in the frame buffer addresses the color map, generating the color programmed at that location.

Typically, the user does not need to read or write pixels directly into the frame buffer. However, for those applications which require direct access, Starbase provides the `gescape` function `R_GET_FRAME_BUFFER` which returns the virtual memory address of the beginning of the frame buffer. This `gescape` is discussed in the appendix. Frame buffer locations are then addressed relative to the returned address. The first byte of the frame buffer (byte 0) represents the upper left corner pixel of the screen. Byte 1 is immediately to its right. Byte 1023 is the last (right-most) pixel on the top line. Byte 1024 is the first (left-most) pixel on the second line from the top. The last (lower right corner) pixel on the screen is byte number 786,431 (767×1024+1023).

Offscreen Memory

The frame buffer is 1024×1024 bytes. The last 256 lines of the frame buffer are not displayed and are referred to as offscreen memory. Offscreen memory may be accessed via the `gescape` function `R_FULL_FRAME_BUFFER` documented in the appendix. Care should be taken when using this `gescape` since other processes, Starbase and window systems, access the frame buffer offscreen memory.

The HP 300H Device Driver allocates a portion of offscreen memory for fill patterns and echo storage. In a raw environment, the first 16 lines are reserved for Starbase fill patterns and each raster echo will use a 64×192 pixel rectangle. In Windows/9000, the first 32 lines are reserved for the Windows/9000 sprite, the last 16 lines are reserved for Starbase fill patterns. In general, the remaining portions of offscreen are allocated from top to bottom. Note: Windows/9000, Fast Alpha and Font Manager also allocate offscreen memory for font storage.

X11 uses offscreen for its sprite, fonts, pixmaps and window backing store (retained rasters). In general, X11 uses offscreen memory intensively; therefore, usage of offscreen memory while running X11 is not recommended.

After reading this chapter, refer to the "Windows/9000 Device Driver" section to find out how this device driver can be used with Windows/9000. Refer to the *Starbase Programming with X11* manual for information on how this device driver can be used with X11.

HP 98549A and HP 319C+

The HP 98549A and HP 319C+ display may also be accessed using the HP 98550 driver. It has higher performance than the HP 300H driver. Applications using the HP 300H driver should not be run simultaneously with applications using the HP 98550 driver on the same display nor should the HP 300H driver be used in an X11 window. The drivers manage offscreen frame buffer memory differently and will interfere with each other. This also applies to the HP Windows/9000 window manager and the X11 server which will (by default) use the HP 98550 driver. However, the Windows/9000 window manager can be directed to use the HP 300H driver by setting environmental variable *WMDRIVER* to `hp300h`. The X11 server cannot be told to use the HP 300H driver. Note also that the HP 98549A and HP 319C+ display is only supported by the HP 300H driver when it is configured as the internal display.

Setting Up the Device

Switch Settings

There are no switches to set on the video boards for these devices. However, when these video boards are used with the HP 310 Processor Board, the display disable switch on the processor board must be set. Look at the four switch group near the back plate. If the third switch from the back plate is set such that the dot closest to the display board's edge is down, the internal display is disabled. Refer to the *Upgrade Video Output Board* Installation Note (HP Part Number 98547-90600) for more details.

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 details. 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.

The mknod parameters are character device with a major number of 12 and a minor number of 0. 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 crt. The following example will create a special device file for this device. Remember that you must be superuser or root to use the mknod command. Note that the leading 0x causes the number to be interpreted hexadecimally.

```
mknod /dev/crt  c  12  0x000000
```

Linking the Driver

The HP 300H Device Driver is located in the /usr/lib directory with the file name libdd300h.a. This device driver may be linked to a program using the absolute path name /usr/lib/libdd300h.a or an appropriate relative path name, or by using the -l option -ldd300h. For example, to compile and link a program for use with this driver use:

```
cc example.c -ldd300h -lsb1 -lsb2 -o example
fc example.f -ldd300h -lsb1 -lsb2 -o example
pc example.p -ldd300h -lsb1 -lsb2 -o example
```

depending upon the language being used.

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/crt.

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 is `hp300h` modified to meet the syntax of the programming language used, namely:

`"hp300h"`	*for C.*
`'hp300h'//char(0)`	*for FORTRAN77.*
`'hp300h'`	*for Pascal.*

Mode — The mode control word consisting of several flag bits which are *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.

- `SPOOLED`—Cannot spool raster devices.
- `0` (zero)—Open the device, but do nothing else. The software color map is initialized on monochrome monitors.
- `INIT`—Open and initialize the device as follows:
  1. Frame buffer is cleared to zeros.
  2. The color map is reset to its default values.
  3. The display is enabled for reading and writing.

Syntax Examples

To open and initialize an HP 300H device for output:

For C Programs:

```
fildes = gopen("/dev/crt",OUTDEV,"hp300h",INIT);
```

For FORTRAN77 Programs:

```
fildes = gopen('/dev/crt'//char(0),OUTDEV,'hp300h'//char(0),INIT)
```

For Pascal Programs:

```
fildes = gopen('/dev/crt',OUTDEV,'hp300h',INIT);
```

Special Device Characteristics

For device coordinate operations, location $(0,0)$ is the upper-left corner of the screen with X-axis values increasing to the right and Y-axis values increasing down. The lower-right corner of the display is $(1023,767)$.

Device Defaults

Number of Color Planes

When the `gopen` procedure is called, this driver asks the device for the number of color planes available. This number can be either 1, 4 or 6. The device driver then acts accordingly.

Dither Default

The default number of colors searched for in a dither cell is 2. The number of colors allowed in a dither cell is 1, 2, 4, 8 or 16.

Raster Echo Default

The default raster echo is the 8×8 array:

```
15  15  15  15  0   0   0   0
15  15  0   0   0   0   0   0
15  0   15  0   0   0   0   0
15  0   0   15  0   0   0   0
0   0   0   0   15  0   0   0
0   0   0   0   0   15  0   0
0   0   0   0   0   0   15  0
0   0   0   0   0   0   0   15
```

The maximum size allowed for a raster echo is 64×64 pixels. The default drawing mode for the raster echo is 7 (a logical *OR*).

Color Plane Defaults

All planes display enabled. All planes write enabled.

Semaphore Default

Semaphore operations are enabled.

Line Type Defaults

The default line types are created with the bit patterns shown below:

Table HP300H-1. Default Line Types

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

Default Color Map

If the fourth gopen parameter is zero (0), the current hardware color map is used on color displays. If the fourth gopen parameter is INIT or RESET_DEVICE, the current color map is initialized to the default values shown below. For a black and white display, only color map indices 0 and 1 are used. For the 4-plane color display, only the indices 0 through 15 are used. For a 6-plane color display, the entire table is used, plus use the `inquire_color_table` procedure to see the remaining 46 colors.

Table HP300H-2. 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

Red, Green and Blue

Each file descriptor opened as an output device has a color table associated with it. If multiple file descriptors are open to the same device, the color table and the device's color map may not always be identical. The color table does not track the color map if the device's color map is changed by another file descriptor path.

For Starbase procedures that have parameters for red, green and blue, it is the color table that is searched for the closest color.

It is usually more efficient to select a color with an index rather than specifying a color with red, blue and green values because of the time it takes for the driver to figure out which pen in the color table most closely matches the specified color.

Selecting a color with the non-index version procedure will allow dithering for filled areas.

Starbase Functionality

Commands Not Supported

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

```
backface_control
bank_switch
bf_control
bf_fill_color
bf_interior_style
bf_perimeter_color
bf_perimeter_repeat_length
bf_perimeter_type
bf_surface_coefficients
bf_surface_model
define_trimming_curve
depth_cue
depth_cue_color
depth_cue_range
hidden_surface
interior_style (INT_OUTLINE)
interior_style (INT_POINT)
light_ambient
light_attenuation
light_model
light_source
light_switch
shade_range
surface_model
surface_coefficients
viewpoint
zbuffer_switch
```

Commands Conditionally Supported

The following commands are supported under the listed conditions:

`block_read`, `block_write`	Note: When using `raw` mode, be careful not to do a `block_read` or `block_write` outside the devices limits.

	When using `raw` mode without using the `R_BIT_MODE gescape`, no clipping is performed. See the `R_BIT_MODE gescape` in the appendix of this manual for more information.
`define_color_table`	On black and white devices, this command defines a software color map, since there is no hardware color map.
`inquire_color_table`	On black and white devices, this command returns the software color map values.
`shade_mode`	The color map mode may be selected but shading can not be turned on.
`text_precision`	Only `STROKE_TEXT` precision is supported.
`vertex_format`	The `use` parameter must be zero; any extra coordinates supplied will be ignored.

Fast Alpha and Font Manager Functionality

This device driver supports raster text calls from the fast alpha and font manager libraries. See the *Fast Alpha/Font Manager's Programmer's Manual* for further information.

Parameters for gescape

The following gescape functions are common to many of the Hewlett-Packard displays supported by Starbase. Detailed information about these functions can be found in Appendix A.

- SWITCH_SEMAPHORE—Semaphore control.
- READ_COLOR_MAP—Read color map.
- BLINK_PLANES—Blink display (blink rate is 2.4 Hz for this device). Not supported in an X11 window.
- R_GET_FRAME_BUFFER—Read frame buffer address.
- R_GET_WINDOW_INFO—Returns frame buffer address of Windows/9000 window.
- R_FULL_FRAME_BUFFER—Full frame buffer.
- R_LOCK_DEVICE—Lock device.
- R_UNLOCK_DEVICE—Unlock device.
- R_BIT_MODE—Bit mode.
- R_BIT_MASK—Bit mask.
- R_DEF_FILL_PAT—Define fill pattern.

Performance Tips

Horizontal and vertical lines are faster than diagonal lines on these devices since the hardware block mover is used to generate the pixels. The procedure block_move is faster then block_read or block_write since the hardware frame buffer block mover can be used.

Cautions

The following cautions are provided in using this driver:

1. As mentioned previously, accessing the off-screen portion of the frame buffer (using `gescape`) should be done with care, since other processes access this region.

2. Certain `gescape` functions should be used with caution since they bypass protection mechanisms used to prevent multiple processes from interfering with each other. For example, since the hardware resources can only be rationally used by one graphics process at a time, the driver sets a semaphore and locks the device before doing any output. This ensures, for example, that process A will not change the replacement rule while process B is in the middle of filling a polygon. It also prevents the terminal (`tty`) driver from overwriting any graphics processes that are outputting to the device. The driver unlocks the device when done processing output. Some of the `gescape` functions listed in this chapter allow the user to change this locking mechanism and should be used with *great caution.*

Contents

HP300L

The HP 300L Device Driver

Device Description

This device driver is used with Hewlett-Packard Series 300 medium-resolution display systems. See the table in the introduction for supported configurations.

The HP 310 processor board has a built-in medium-resolution monochrome display system. The HP 98542A and HP 98543A video boards fit in an SPU system slot. The monochrome display systems have a single plane of frame buffer. The color display system has four planes of frame buffer to provide 16 simultaneous colors. A color map provides eight bits per color (for red, green and blue), providing a color palette of over 16 million colors.

All three systems have a resolution of 1024×400 pixels; however, this driver treats the display systems as having a resolution of 512×400 pixels since each pixel in the frame buffer has an aspect ratio of 2 to 1. By writing two pixels for every dot to be displayed, square pixels are produced. Some applications or subsystems such as HP Windows/9000 may use the higher resolution. The `gescape` function `TC_HALF_PIXEL`, documented later in this section, can be used to allow `block_read` and `block_write` access to the full resolution.

These display systems are bit-mapped devices with special hardware for:

- Write enabling planes.
- Displaying planes.
- Writing pixels to the frame buffer with a given replacement rule (see `drawing_mode`).
- Blinking planes.
- Moving a block of pixels from one place in the frame buffer to another.

Both the monochrome and color displays are organized as an array of bytes, with each byte representing a pixel on the display. For the monochrome display, the Least Significant Bit (LSB) of each byte controls the display, with 0 for black (pixel off) and 1 for white (pixel on).

For the color display, the four LSBs of each byte determine the color, providing color values from 0-15. These values are used to address the color map. The color map is basically a RAM table that has 16 addressable locations and is 24 bits wide (eight bits each for red, green and blue). Thus, the pixel value in the frame buffer addresses the color map, generating the color programmed at that location.

Typically, the user does not need to directly read or write pixels in the frame buffer. However, for those applications which require direct access, Starbase does provide the `gescape` function `R_GET_FRAME_BUFFER`, which returns the virtual memory address of the beginning of the frame buffer. This `gescape` is discussed in the appendix. Frame buffer locations are then addressed relative to the returned address. The first byte of the frame buffer (byte 0) represents the upper left corner pixel of the screen. Byte 1 is immediately to its right. Byte 1023 is the last (right-most) pixel on the top line. Byte 1024 is the first (left-most) pixel on the second line from the top. The last (lower right corner) pixel on the screen is therefore byte number 409599 ($399 \times 1024 + 1023$).

Offscreen Memory

The frame buffer is 1024×512 bytes. The last 112 lines of the frame buffer are not displayed and are referred to as offscreen memory. Offscreen memory may be accessed via the `gescape` function `R_FULL_FRAME_BUFFER` documented in the appendix. Care should be taken when using this `gescape` since other processes, Starbase and window systems, access the frame buffer offscreen memory.

The HP 300L Device Driver allocates a portion of offscreen memory for fill patterns and echo storage. In a raw environment, the first 16 lines are reserved for Starbase fill patterns, and each raster echo will use a 64×384 byte rectangle (64×192 square pixels). In Windows/9000, the first 32 lines are reserved for the Windows/9000 sprite, the last 16 lines are reserved for Starbase fill patterns. In general, the remaining portions of offscreen are allocated from top to bottom. Note: Window/9000, Fast Alpha and Font Manager also allocate offscreen memory for font storage.

X11 uses offscreen for its sprite, fonts, pixmaps and window backing store (retained rasters). In general, X11 uses offscreen memory very intensively; therefore, usage of offscreen memory while running X11 is not recommended.

After reading this section, refer to the section "Windows/9000 Device Driver" to find out how this device driver can be used with Windows/9000. Refer to the *Starbase Programming with X11* manual for more information on how this device driver can be used with X11.

Setting Up the Device

Switch Settings

There are no switches to set on the video boards for these devices. However, when the HP 98542A or HP 98543A video boards are used with the HP 310 processor board, the display disable switch on the processor must be set. Look at the four switch group near the back plate. If the third switch from the back plate is set such that the dot closest to the display board's edge is down, the internal display is disabled. Refer to the *Upgrade Video Output Board* Installation Note (HP Part Number 5958-4342) for more details.

Special Device Files (mknod)

The `mknod` command (see `mknod(8)` man page), creates a special device file which is used to communicate between the computer and the peripheral device. 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.

The `mknod` parameters are character device with a major number of 12 and a minor number of 0. 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 **crt**. The following example will create a special device file for this device. Remember that you must be superuser or root to use the `mknod` command. Note that the leading `0x` causes the number to be interpreted hexadecimally.

```
mknod /dev/crt  c  12  0x000000
```

Linking the Driver

The HP 300L Device Driver is located in the `/usr/lib` directory with the file name `libdd300l.a`. This device driver may be linked to a program using the absolute path name `/usr/lib/libdd300l.a`, an appropriate relative path name, or the `-l` option `-ldd300l`. For example: to compile and link a program for use with this driver, use:

```
cc example.c -ldd300l -lsb1 -lsb2 -o example
fc example.f -ldd300l -lsb1 -lsb2 -o example
pc example.p -ldd300l -lsb1 -lsb2 -o example
```

depending upon the language being used.

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/crt`.

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 is `hp300l` modified to meet the syntax of the programming language used, namely:

`"hp300l"`	*for C.*
`'hp300l'//char(0)`	*for FORTRAN77.*
`'hp300l'`	*for Pascal.*

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.

- SPOOLED—cannot spool raster devices.

- 0—open the device, but do nothing else. The software color map is initialized on monochrome monitors.
- INIT—open and initialize the device as follows:
    1. Frame buffer is cleared to 0s.
    2. The color map is reset to its default values.
    3. The display is enabled for reading and writing.

Syntax Examples

To open and initialize an HP 300L device for output:

For C Programs:

```
fildes = gopen("/dev/crt",OUTDEV,"hp300l",INIT);
```

For FORTRAN77 Programs:

```
fildes = gopen('/dev/crt'//char(0), OUTDEV,'hp300l'//char(0),INIT)
```

For Pascal Programs:

```
fildes = gopen('/dev/crt',OUTDEV,'hp300l',INIT);
```

Special Device Characteristics

For device coordinate operations, location $(0,0)$ is the upper-left corner of the screen with X-axis values increasing to the right and Y-axis values increasing down. The lower-right corner of the display is $(511,399)$.

Device Defaults

Number of Color Planes

When the `gopen` procedure is called, this driver asks the device for the number of color planes available. This number can be either 1 or 4. The device driver then acts accordingly.

Dither Default

The default number of colors searched for in a dither cell is 2. The number of colors allowed in a dither cell is 1, 2, 4, 8 or 16.

Raster Echo Default

The default raster echo is the 8×8 array:

15	15	15	15	0	0	0	0
15	15	0	0	0	0	0	0
15	0	15	0	0	0	0	0
15	0	0	15	0	0	0	0
0	0	0	0	15	0	0	0
0	0	0	0	0	15	0	0
0	0	0	0	0	0	15	0
0	0	0	0	0	0	0	15

The maximum size allowed for a raster echo is 64×64 pixels. The default drawing mode for the raster echo is 7 (a logical *OR*).

Color Planes Defaults

All planes display enabled. All planes write enabled.

Semaphore Default

Semaphore operations are enabled.

Line Type Defaults

The default line types are created with the bit patterns as shown in the following table.

Table HP300L–1.

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

Default Color Map

If the fourth gopen parameter is zero (0), the current hardware color map is used on color displays. If the fourth gopen parameter is INIT or RESET_DEVICE, the current color map is initialized to the default values shown below. For a black and white display, only color map indices 0 and 1 are used.

Table HP300L-2. 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

Red, Green and Blue

Each file descriptor opened as an output device has a color table associated with it. If multiple file descriptors are open to the same device, the color table and the device's color map may not always be identical. The color table does not track the color map if the device's color map is changed by another file descriptor path. For Starbase procedures having parameters for red, green and blue, it is the color table that is searched for the closest color.

It is usually better (more efficient) to select a color with an index rather than specifying a color with red, blue and green values due to the time it takes for

the driver to figure out which pen in the color table most closely matches the specified color.

Selecting a color with the non-index version procedure will allow dithering for filled areas, desirable in some cases.

Starbase Functionality

Commands Not Supported

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

```
backface_control
bank_switch
bf_control
bf_fill_color
bf_interior_style
bf_perimeter_color
bf_perimeter_repeat_length
bf_perimeter_type
bf_surface_coefficients
bf_surface_model
define_trimming_curve
depth_cue
depth_cue_color
depth_cue_range
hidden_surface
interior_style (INT_OUTLINE)
interior_style (INT_POINT)
light_ambient
light_attenuation
light_model
light_source
light_switch
shade_range
surface_model
surface_coefficients
viewpoint
zbuffer_switch
```

Commands Conditionally Supported

The following commands are supported under the listed conditions:

`block_read`, `block_write`	Note: When using `raw` mode, be careful not to do a `block_read` or `block_write` outside the device's limits. When using `raw` mode without the `R_BIT_MODE gescape`, no clipping is performed. See the `R_BIT_MODE gescape` in the appendix of this manual for more information.
`define_color_table`	Since there is no hardware color map on black and white devices, this command defines a software color map.
`inquire_color_table`	On black and white devices, this command returns the software color map values.
`text_precision`	Only `STROKE_TEXT` precision is supported.
`shade_mode`	The color map mode may be selected, but shading can not be turned on.
`vertex_format`	The `use` parameter must be zero; any extra coordinates supplied will be ignored.

Fast Alpha and Font Manager Functionality

This device driver supports raster text calls from the fast alpha and font manager libraries. See the *Fast Alpha/Font Manager's Programmer's Manual* for further information.

Parameters for gescape

The following gescape functions are common to many of the Hewlett-Packard displays supported by Starbase. Detailed information about these functions can be found in the appendix.

- SWITCH_SEMAPHORE—Semaphore control.
- READ_COLOR_MAP—Read color map.
- BLINK_PLANES—Blink display (blink rate is 2.4 Hz for this device.) Not supported in an X11 window.
- R_GET_FRAME_BUFFER—Read frame buffer address.
- R_GET_WINDOW_INFO—Returns frame buffer address of Window/9000 window.
- R_FULL_FRAME_BUFFER—Full frame buffer.
- R_LOCK_DEVICE—Lock device
- R_UNLOCK_DEVICE—Unlock device.
- R_BIT_MODE—Bit mode.
- R_BIT_MASK—Bit mask.
- R_DEF_FILL_PAT—Define fill pattern.

The gescape function TC_HALF_PIXEL is unique to this driver and is presented next in this section.

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

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

FORTRAN77 Syntax

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

Pascal Syntax

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

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

Performance Tips

Horizontal and vertical lines are faster then diagonal lines on these devices since the hardware block mover is used to generate the pixels. The procedure `block_move` is faster then `block_read` or `block_write` since the hardware frame buffer block mover can be used.

Cautions

The following cautions are provided in using this driver:

1. As mentioned previously, accessing the off-screen portion of the frame buffer (using the `gescape` function) should be done with care since other processes access this region.
2. Certain `gescape` functions should be used with caution since they bypass protection mechanisms used to prevent multiple processes from interfering with each other. For example, since the hardware resources can only be rationally used by one graphics process at a time, the driver activates a semaphore and locks the device before doing any output. This ensures, for example, that process A will not change the replacement rule while process B is in the middle of filling a polygon. It also prevents the terminal (`tty`) driver from overwriting any graphics processes that are outputting to the device. The driver unlocks the device when finished processing output. Some of the `gescape` functions listed in the appendix allow the user to change this locking mechanism but should be used with *great caution.*

Contents

HP9836A

The HP 9836A Device Driver

Device Description

This device driver is used to provide graphics output on the Series 300 HP 98546A Display Card. This display is a bit-mapped device which has a resolution of 512×390 pixels with a single plane frame buffer. A separate non-bit mapped alpha plane allows text and graphics to be manipulated independently.

This display card provides compatibility with programs written for HP 9836A Series 200 models and for HP 98204B displays.

The interface for this device plugs into an I/O slot of supported SPUs. See table 1-8 in the introduction of this manual for the SPUs which support this device.

The display is organized as an array of bytes, with each byte representing 8 pixels on the display. Typically, the user does not need to directly read or write pixels in the frame buffer. However, for those applications which require direct access, Starbase provides the `gescape R_GET_FRAME_BUFFER`, which returns the virtual memory address of the beginning of the frame buffer (this `gescape` is discussed in the appendix).

Frame buffer locations are addressed relative to the returned address. The first byte of the frame buffer (byte 0) represents eight pixels in the upper left corner of the screen. The Most Significant Bit (MSB) of that byte holds the pixel in the upper left corner of the display. The Least Significant Bit (LSB) of the first byte holds the right-most pixel in the byte (that is, pixel number 8). Byte 1 is immediately to its right. Byte 63 holds the last 8 pixels on the top line. Byte 64 hold the 8 pixels below the first line. The last (lower right corner) set of 8 pixels on the screen is in byte number 24,959 (389×64+63).

This display does not support Windows/9000 or the XWindows system.

Setting Up the Device

Switch Settings

For normal operation of this device, there are no switches to set.

Special Device Files (mknod)

The `mknod` command (see `mknod` in the man pages), creates a special device file which is used to communicate between the computer and the peripheral device. 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.

The `mknod` parameters are the character device with a major number of 12 and a minor number of 0. 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 `crt`. The following example will create a special device file for this device. Remember that you must be superuser or root to use the `mknod` command.

```
mknod /dev/crt  c  12  0x000000
```

Linking the Driver

The HP 9836A Device Driver is located in the `/usr/lib` directory with the file name `libdd9836a.a`. This device driver may be linked to a program using the absolute path name `/usr/lib/libdd9836a.a`, an appropriate relative path name, or the `-l` option `-ldd9836a`. For example: to compile and link a program for use with this driver, use:

```
cc example.c -ldd9836a -lsb1 -lsb2 -o example

fc example.f -ldd9836a -lsb1 -lsb2 -o example

pc example.p -ldd9836a -lsb1 -lsb2 -o example
```

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/crt.

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 is hp9836a, hp98546a, or hp98204b modified to meet the syntax of the programming language used, namely:

"hp9836a"	*for C.*
'hp9836a'//char(0)	*for Fortran77.*
'hp9836a'	*for Pascal.*

Mode — The mode control word consists of several flag bits which are *or* ed together. Listed below are those those flag bits which have device-dependent actions. SPOOLED flag bits have no affect for this driver. Those flags not discussed below operate as defined by the gopen procedure.

- SPOOLED—cannot spool raster devices.
- 0—open the device and initialize the software color map
- INIT—open and initialize the device as follows:
  1. Frame buffer is cleared to 0s.
  2. The color map is set to its default values.
  3. The display is enabled for reading and writing.

Syntax Examples

To open and initialize an HP 9836A device for output:

C Syntax Examples

```
fildes = gopen("/dev/crt",OUTDEV,"hp9836a",INIT);
```

FORTRAN77 Syntax Examples

```
fildes = gopen('/dev/crt'//char(0), OUTDEV,'hp9836a'//char(0),INIT)
```

Pascal Syntax Examples

```
fildes = gopen('/dev/crt',OUTDEV,'hp9836a',INIT);
```

Special Device Characteristics

For device coordinate operations, location $(0,0)$ is the upper-left corner of the screen with X-axis values increasing to the right and Y-axis values increasing down. The lower-right corner of the display is therefore $(511,389)$.

Device Defaults

Number of Color Planes

This display supports one frame buffer plane.

Dither Default

The default number of colors searched for in a dither cell is 2. The number of colors allowed in a dither cell is 1, 2, 4, 8 or 16.

Raster Echo Default

The default raster echo is the 8×8 array:

```
1 1 1 1 0 0 0 0
1 1 0 0 0 0 0 0
1 0 1 0 0 0 0 0
1 0 0 1 0 0 0 0
0 0 0 0 1 0 0 0
0 0 0 0 0 1 0 0
0 0 0 0 0 0 1 0
0 0 0 0 0 0 0 1
```

The maximum size allowed for a raster echo is 64×64 pixels. The default drawing mode for the raster echo is 7 (a logical *OR*).

Semaphore Default

Semaphore operations are enabled.

Line Type Defaults

The default line types are created with the bit patterns shown. The Starbase default line type is `SOLID`, line type `0`.

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

Default Color Map

Table HP9836A-1. HP9836A Default Color Table

Index	Color	Red	Green	Blue
0	Black	0.0	0.0	0.0
1	White	1.0	1.0	1.0

Starbase Functionality

Exceptions to Standard Starbase Support

The following commands are supported under the listed conditions:

`block_read, block_write` — When the `raw` parameter is set to `TRUE`, it indicates that `DATA` is arranged with 8 pixels/byte. The data rounds to a pixel in the X-axis direction that aligns with a byte or word boundary. Clipping of the data is not performed in `raw` mode.

`define_color_table`	Since there is no hardware color map, this command defines a software color map on black and white devices.
`inquire_color_table`	This command returns the software color map values on black and white devices.
`interior_style`	Only the `INT_SOLID`, `INT_HATCH`, and `INT_HOLLOW` styles are supported.
`text_precision`	Only `STROKE_TEXT` precision is supported.
`await_retrace`	This routine has no effect on this display.

Commands Not Supported

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

```
bank_switch                  depth_cue_range
backface_control             hidden_surface
bf_control                   light_ambient
bf_fill_color                light_attenuation
bf_interior_style            light_model
bf_perimeter_color           light_source
bf_perimeter_repeat_length   light_switch
bf_perimeter_type            shade_range
bf_surface_coefficients      surface_model
bf_surface_model             surface_coefficients
define_trimming_curve        viewpoint
depth_cue                    zbuffer_switch
depth_cue_color
```

Contents

HP98550A

The HP 98550A Device Driver

Device Description

This device driver is used with Series 300 and Series 800 systems. See the table in the introduction for supported configurations.

The HP 98548A display is supported only on some Series 300 SPUs. This display has a resolution of 1280×1024 pixels. It provides a single frame buffer plane.

The HP 98549A Color Display Board is supported only on some Series 300 SPUs. It has a resolution of 1024×768 pixels and 6-color planes. The color planes can also be soft-configured as four image planes and two overlay planes.

The HP 319C is functionally equivalent to the HP 98549A display.

The high resolution HP 98550A display is supported on both Series 300 and Series 800 SPUs. This display has 1280×1024 pixels. It has eight color planes for 256 colors, plus two full-time overlay planes.

In this color system, a color map provides eight bits per color (for red, green and blue), providing a color palette of over 16 million colors.

These display boards fit in a Series 300 SPU system slot, or in a Series 800 A1020A Bus Converter. They are bit-mapped devices with special hardware for:

- Generating vectors.
- Filling polygons.
- Write enabling planes.
- Displaying planes.
- Writing pixels to the frame buffer with a given replacement rule.
- Blinking planes.
- Moving a block of pixels from one place in the frame buffer to another.

- Pixel replication.
- Three-operand raster combinations with a 16×16 tiling mask.
- Independent overlay planes with transparency.

Note Series 300 Only

The HP 98549A display board and HP 319C display may also be accessed by the HP 300H driver, but only if it is configured as the internal display. The HP 300H driver provides lower performance than the HP 98550 driver (in most operations). Applications using the HP 300H driver should not be run simultaneously with applications using the HP 98550 driver on the same display. The drivers manage offscreen frame buffer memory differently and will interfere with each other. This also applies to the HP Windows/9000 window manager which will (by default) use the HP 98550 driver. However, it can be directed to use a different driver by setting environment variable `WMDRIVER` to that driver. This also applies to the X Window System, which automatically uses the hardware in a way compatible with the HP 98550 driver, but not with the HP300H driver.

Overlay Planes and Image Planes

The color displays supported by this driver have both image and overlay planes.

There are two overlay planes in the HP 98550A system, and two planes of the HP 98549A or HP 319C may be configured by software as overlay planes. The overlay planes are *in front of* the image planes. An independent 4-entry color map is provided for the overlay planes. Denote the four colors provided by the overlay planes as colors 0, 1, 2 and 3. Colors 1, 2 and 3 are dominant and are always displayed for the corresponding pixels, regardless of what is in the image planes. Color 0, however, may be made "transparent," allowing the image planes to show through. For example, a cursor might be drawn in the overlay planes with color 1. If color zero is transparent and specified everywhere else in the overlay planes, the cursor will be dominant but independent of any image in image planes.

This is a fast way of drawing cursors because a cursor drawn directly in the image planes must be removed before drawing a new object. The driver may avoid these cursor-remove and -replace steps if the cursor is in the overlay planes. The gescape, R_OVERLAY_ECHO, can be used to control cursor placement.

A gescape, GR2D_OVERLAY_TRANSPARENT, may be used to make overlay color 0 dominant. This will cause all 4-overlay plane colors to be displayed and the image planes to be entirely obscured. The default mode is transparent.

Use of the gescape is not recommended when running in a window environment.

There are 4-, 6-, or 8-image planes. The image planes index the main color map.

Interactions with the ITE

The Internal Terminal Emulator (ITE) operates in the image planes but not the overlay planes. There may be interactions if graphics and the ITE are active simultaneously in the image planes. The ITE treats the HP 98549A and HP 319C as 6-plane devices.

On all displays, a hard ITE reset (shift-control-reset) will clear all planes. This hard reset operation will also clear up any bad hardware states that may occur if a graphics process is aborted.

Windows Operation in the Overlay and Image Planes

The HP Windows/9000 system is only supported in the image planes, not in the overlay planes. Windows in the image planes behave in the usual way. The overlay planes must be transparent to see windows (or any data) in the image planes. Starbase overlay planes are not supported when in HP Windows/9000 windows.

Note Windows/9000 is supported only on Series 300 computers.

The HP Windows/9000 system operates in all six planes of the HP 98549A and HP 319C display by default.

The X Window System is supported in various configurations and modes on the devices supported by the HP 98550 driver. See the *Starbase Programming with X11* manual for a complete discussion of X Window support.

Windows with and without Retained Rasters

Consider a graphics window that is partially obscured by another window. What happens when you try to draw graphics to the part of the window that is obscured? There are two options:

- Draw only to the visible parts of the window and ignore any parts that are obscured. In HP Windows/9000, an application that does this will typically monitor the `SIGWINDOW` signal and repaint the entire image when obscured parts of a window are made visible. See the HP *Windows/9000 Programmer's Manual* and `signal`(2) in the *HP-UX Reference* manual. In X Windows System, an application will receive and handle exposure events.
- For those parts of the window that are obscured, draw the image to *memory* instead of the frame buffer. When the affected portion of the window is made visible (unobscured), the window system updates the display with the appropriate graphical data from memory. Graphical data stored (retained) in memory is called a **retained raster** or **backing store.**

HP Windows/9000 and the X Window System support both options. Support for retained rasters is provided by `/usr/lib/libddbyte.a` (called the byte driver). The graphics window must be created with a retained raster. Linking the byte driver allows Starbase to draw images to memory for obscured parts of the window and allows the window system to update the screen from memory if previously obscured parts of the window are made visible. Review the "HP Windows/9000 Device Driver" chapter of this manual for more information about retained rasters. See the *Starbase Programming with X11* manual for more information on retained rasters in X Windows.

The `/usr/lib/libddbit.a` (bit driver) is supported on the HP 98548A only.

Retained Raster Support

In general, those Starbase operations that draw to the display are also supported as a retained raster by the byte driver (`/usr/lib/libddbyte.a`). There are, however, some exceptions that you should be aware of. These exceptions all involve use of the `gescape` operation to access device-dependent features. When the `gescape` operations listed below are used with a retained graphics window, they will have the desired effect for the visible portion of the window but may cause the retained raster for obscured parts to be altered in inconsistent ways. The features involved (along with the names of the affected `gescape` operations) are listed below. For more details on the gescape operations, refer to later sections in this chapter.

Note Because the `gescape` operations are device-dependent, the exceptions discussed below may be removed in future drivers. Also, if the exceptions to retained raster support discussed below prove troublesome in your application, it is recommended that you consider not using retained rasters but instead detect window events and repaint the window when a previously obscured portion of a window is made visible. See the section "Event Detection" in the *Windows/9000 Programmer's Manual* for more information about HP Windows/9000.

Multiple-Plane Bit/Pixel Support

When `block_read` or `block_write` is used with the `raw` parameter `TRUE`, and `raw` mode is enabled by the `R_BIT_MODE gescape`, the driver supports bit/pixel frame buffer access to single planes.

The `gescape` operation `R_BIT_MASK` defines a plane mask to the driver and is used for bit/pixel access to a single plane in the frame buffer. As in other device drivers, only the plane corresponding to the highest bit set in the mask is transferred. This `gescape` is supported for retained rasters; i.e. the correct data is returned from the retained raster for those parts of the window that are obscured.

The `gescape` operation `GR2D_PLANE_MASK` defines a mask that allows multiple planes to be read or written. This `gescape` is not supported in retained rasters. It returns the correct data from the visible portions, but not from the obscured portions.

Pixel Replication

`GR2D_REPLICATE` triggers operations to do pixel replication in the frame buffer. This is a complicated operation involving multiple hardware `block_moves`; the retained raster will not be affected.

16×16 Fill Pattern

`GR2D_FILL_PATTERN` sets a 16×16 fill pattern used by the driver as a source to fill polygons until the fill pattern is redefined, either by a call to `fill_color` or `fill_color_index` or to the `gescape` `R_DEF_FILL_PAT` or `GR2D_FILL_PATTERN`. The retained raster only supports the `R_DEF_FILL_PAT` 4×4 pattern. The `pattern_define` routine for Starbase is recommended instead of this `gescape`.

Note For a discussion of replacement rules, review the "Drawing Modes" section of the "Frame Buffer Control & Raster Operations" chapter of the *Starbase Graphics Techniques* manual.

There are several `gescape` operations associated with the use of three-operand raster support:

- `GR2D_DEF_MASK`
- `GR2D_MASK_ENABLE`
- `GR2D_MASK_RULE`

Since these `gescape` operations alter the rule and pattern used for `block_write` and `block_move`, the retained raster will be affected during later raster operations and the results will not be consistent with what appears on the screen.

Three-Operand Raster Operations

The Starbase `drawing_mode` procedure defines a set of **replacement rules** (**drawing modes**) that may be used when primitives are drawn to the screen. Each of these rules represents one of the sixteen possible truth tables that may be generated from two logical operands, and defines the results that will be obtained when combining a source bit and a destination bit. The resulting bit becomes the new value in the frame buffer. For example, the most often-used rule is `SOURCE`, that is, the value to the source bit dominates, and is defined by the following truth table:

Table HP98550A-1.

Source	Destination†	Result‡
0	0	0
0	1	0
1	0	1
1	1	1

† Destination means current frame buffer content.

‡ Result means final frame buffer content.

Three logical operands may be combined to generated 256 different **three-operand rules.** The third operand will be referred to as the "mask" operand, and although it does not have to be considered a logical mask, it is often used that way in common applications. Some literature refers to the third-operand as a *pattern* operand, but we will avoid this to prevent confusion with the "dither pattern" or "fill pattern" that is used as a source operand during polygon fill.

Shown below is the truth table for the rule "use $\langle source \rangle$ if $\langle mask \rangle = 1$, keep $\langle destination \rangle$ if $\langle mask \rangle = 0$."

Table HP98550A-2.

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

† Destination means current frame buffer content.

‡ Result means final frame buffer content.

When filling or copying an area of the frame buffer, a source **region** is combined with a destination region according to the current drawing mode. This means that each bit of each pixel from the source region is combined with the corresponding

bit of the corresponding pixel of the destination according to the drawing mode. The most general form of a three-operand capability would involve a third region representing the mask operand.

The supported displays provide hardware support for three-operand drawing modes, but there is a restriction on the mask operand: it is defined as a 16×16-pixel rectangle that is repeated across the entire raster in both X and Y, starting at the upper left corner (that is, the mask **tiles** the raster). For example, this restriction precludes the use of the mask operand to trim the edges of a large irregular figure, but it is convenient in imposing a pattern on the source during a `block_move` or `block_write` operation. Hardware tiling is tied to device coordinates, so the same primitive drawn to different locations on the screen may not appear exactly the same.

For example, suppose a raster rectangle is to be copied to another area on the screen using `block_move`. In normal, two-operand mode, the source data is applied (unchanged) to the destination area according to the current two-operand drawing mode. In three-operand mode, however, the mask operand could be defined as a checkerboard that truly masks the source, producing a checkerboard effect in combining source and destination. The following figure shows how this might look:

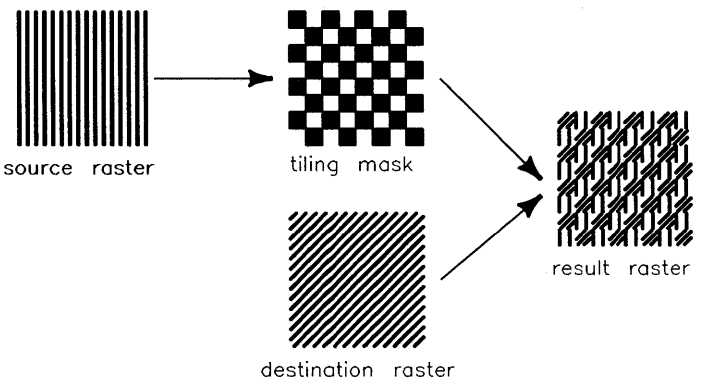


Figure HP98550A-1. Three-Operand Mode Dithering

Access to the hardware capability is provided through three `gescape` operations:

- `GR2D_DEFINE_MASK` defines the 16×16 mask operand
- `GR2D_MASK_RULE` defines the three-operand rule

- GR2D_MASK_ENABLE enables or disables the three-operand combination

When the feature is disabled, the current two-operand rule is in effect. When enabled, the mask operand and three-operand rule may be applied to `block_write` and `block_move` operations, or to these plus raster text operations.

Frame Buffer Access

The supported displays provide hardware support for frame buffer access in both plane-major and pixel-major modes. Pixel-major access views the frame buffer as an array of bytes, one byte per pixel, with `n` significant bits each byte. The normal operation of Starbase procedure `block_read` and `block_write` treat the frame buffer in pixel-major mode. Plane-major mode addresses the frame buffer as a set of *n* planes, each consisting of a packed array of bits, one bit per pixel. This may be used, for example, to transfer data quickly from one plane to another or from a data array to a plane. In this driver, plane-major access may be made with the `block_read` and `block_write` procedure by setting the `raw` parameter to `TRUE` and enabling `raw` mode using the `R_BIT_MODE gescape`. (Not all drivers provide this capability.) Details of this form of access are provided below in the section entitled "Starbase Functionality."

Starbase support for double-buffering by planes may be used to aid in smooth animation. This is done using `write_enable` and `display_enable` masks; there is no direct hardware support for double buffering.

Note Series 800 Dependency

When writing to IO space, accesses must be on word (32 bit) boundaries. The frame buffer is mapped as one word per pixel. Therefore, pixels should be addressed on word boundaries when directly accessing the frame buffer. Also, the pixel value is in the least significant byte of the word.

HP 98548A Display

The HP 98548A display has a resolution of 1280×1024 pixels and provides 1-image plane. There are no overlay planes.

Creation of a special device file is discussed in the next section.

The physical frame buffer is 2048×1024 bytes. The last 768 bytes of each line of the frame buffer (to the "right" of the screen) are not displayed and are used for cursor, font and working storage.

The first byte (byte 0) of the frame buffer represents the upper left corner pixel of the screen. Byte 1 is immediately to its right. Byte 1279 is the (right-most) pixel on the top line. The next 768 bytes are not displayable. Byte 2048 is the first (left-most) pixel on the second line from the top. The last (lower right corner) pixel on the screen is byte number 2,096,383 (1023×2048+1279).

For normal `block_read` and `block_write` operations, the data is in the least significant bit of each byte.

HP 98549A and HP 319C Displays

The HP 98549A display board and HP 319C display have a resolution of 1024×768 pixels and provide six planes of frame buffer. These six planes of frame buffer may be configured by the driver in one of two modes:

- One logical device that has a full six planes and provides 64 simultaneous colors. This is referred to as "6-plane mode".
- Two logical devices, providing 4-image planes (16 simultaneous colors) and 2-overlay planes (four simultaneous colors). This is referred to as "4+2-plane mode".

The HP 98549A color display is supported by the `hp98550` and `hp98556` device drivers. It may only be used with the Series 300 Models 319, 320, 330, 350, 360, and 370 SPUs.

Creation of the special device files used to access the device in these modes is discussed below. Selection of the mode is done when a file is opened with the `gopen` procedure.

Note

Do not open an HP 98549A or HP 319C simultaneously in both 6-plane mode and 4+2-plane mode. Doing so will cause indeterminate results. You may simultaneously open the overlay planes and the 4-image planes using two different file descriptors. If graphics are being done in the overlay planes, do not use the `gescape R_OVERLAY_ECHO` to move the cursor from the image planes into the overlay planes since this will interfere with the overlay graphics.

The X Window System only supports the 6-plane mode.

The physical frame buffer is 1024 bytes wide and 1024 bytes high. The last 256 lines of the frame buffer (below the screen) are not displayed and are used for cursor, font, and working storage.

The first byte (byte 0) of the frame buffer represents the upper left corner pixel of the screen. Byte 1 is immediately to its right. Byte 1023 is the last (right-most) pixel on the top line. Byte 1024 is the first (left-most) pixel on the second line of the screen. The last (lower right corner) pixel on the screen is byte number 786,431 (767×1024+1023).

For normal (non-`raw`) `block_read` and `block_write` operations, the data is in the least significant bits of each byte. The number of valid bits depends on the logical device opened (2, 4, or 6 valid bits).

`R_GET_FRAME_BUFFER` returns the virtual memory address of the beginning of the frame buffer. Caution is necessary if the HP 98549A or HP 319C is opened in 4+2-plane mode, because the frame buffer address returned for the 4-image planes is the same as the address for the 2-overlay planes. To ensure that you only access the planes that are opened, `R_LOCK_DEVICE` (using the file descriptor for the appropriate planes) should be used to lock the device before reading or changing the frame buffer. Note that your program must shift each data byte to the left by four bits in order to directly write it to the overlay planes (this shifting is done automatically when `block_write` is used). Use `R_UNLOCK_DEVICE` to unlock the device after the access.

HP 98550A Display

The HP 98550A display has a resolution of 1280×1024 pixels and provides eight image planes, plus two dedicated overlay planes.

Special device files may be created to access:

- The eight image planes.
- The two overlay planes.

Creation of the special device files is discussed below. Selection of the plane access mode is done when one of these files is opened with the **gopen** procedure. The two logical devices may be opened simultaneously, but moving the image planes cursor into the overlay planes may interfere with graphics in the overlay planes.

The physical frame buffer is 2048×1024 bytes. The last 768 bytes of each line of the frame buffer (to the right of the screen) are not displayed and are used for cursor, font and working storage.

The first byte (byte 0) of the frame buffer represents the upper left corner pixel of the screen. Byte 1 is immediately to its right. Byte 1279 is the last (right-most) pixel on the top line. The next 768 bytes are not displayable. Byte 2048 is the first (left-most) pixel on the second line from the top. The last (lower right corner) pixel on the screen is byte number 2,096,383 (1023×2048+1279).

For normal (non-raw) `block_read` and `block_write` operations to the image planes, the data is in all 8 bits of each byte. For the 2-overlay planes, data is in the lower 2 bits of each byte.

Series 800 Dependency

When using `R_GET_FRAME_BUFFER` for direct user access to the frame buffer, correct access can only be assured by using `R_LOCK_DEVICE` and `R_UNLOCK_DEVICE`.

`R_LOCK_DEVICE` should be used just prior to direct frame buffer access.

`R_UNLOCK_DEVICE` should be used directly after the frame buffer access and before any other Starbase commands.

Setting Up the Device

Switch Settings

The board has a bank of eight switches. The default switch settings configure the display as the internal (console) display. If the most significant bit is set, the board is configured in the 32-bit address space (not supported on the Model 320). The first four settings in the 32-bit space are not supported.

Table HP98550A-3. Switch Settings Supported on Series 300

Switches	Select Code	Comment
00000001	Internal	Default
00000010	-	Not Supported
⋮		
01111111	-	Not Supported
10000000	-	Not Supported
⋮		
10000011	-	Not Supported
10000100	132	Supported
10000101	133	Supported
⋮		
11111111	255	Supported

The HP 319C display has no switches and is always at the internal select code.

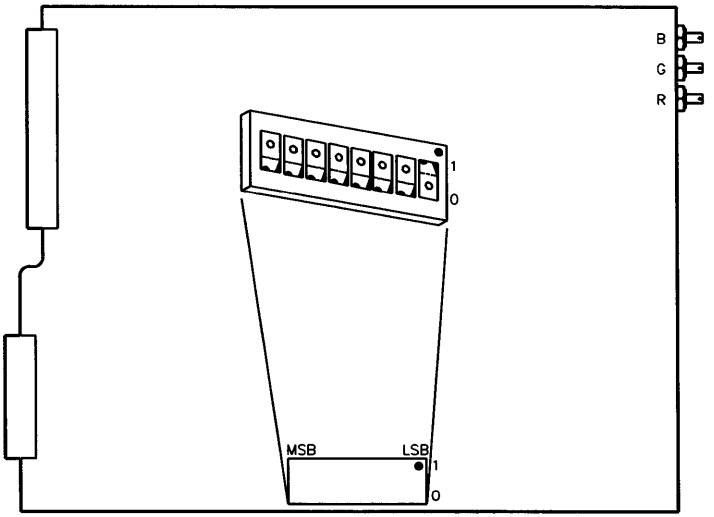


Figure HP98550A–2. Series 300 Default Switch Settings

The only Series 800 configuration supported is with the switches set to the position shown in the following figure.

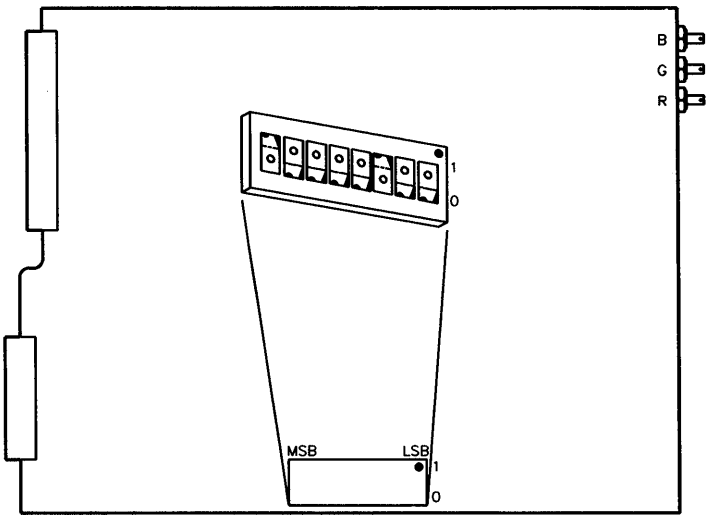


Figure HP98550A-3. Series 800 Default Switch Settings

Special Device Files (mknod)

The `mknod` command (see `mknod(8)` in the *HP-UX Reference* manual), creates a special device file that is used to communicate between the computer and the display device. 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.

The Series 300 `mknod` parameters are:

Character device with a major number equal to 12 and a minor number equal to 0x000000 (internal) or 0x⟨*sc*⟩0200 (externally).

The Series 800 `mknod` parameters are:

Character device with a major number equal to 14 and a minor number of the form 0x00⟨*lu*⟩00 where ⟨*lu*⟩ is the logical unit of the A1020A graphics subsytem. Note, the leading 0x causes the number to be interpreted hexadecimally.

Although special device files may 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 the default device is `crt`.

The normal device file (last digit of the minor number is zero) will cause the HP 98549A or HP 319C driver to open the display in 6-plane mode.

The following example will create a special device file for the Series 300 internal display. Remember that you must be superuser or root to use the `mknod` command.

```
mknod /dev/crt  c  12  0x000000
```

The next example creates a device file for an external configuration:

```
mknod /dev/crt  c  12  0x⟨sc⟩0200
```

The next example will create a special device file for the Series 800. Again, you must be superuser or root to use the `mknod` command.

```
mknod /dev/crt  c  14  0x00⟨lu⟩00
```

To open the driver to the 2-overlay planes, the last digit of the minor number must be one. Similarly, to access only the image planes, the last digit of the minor number must be two. Use of either of these device files will cause the HP 98549A or HP 319C to be placed in the 4+2-plane mode.

For the HP 98548A, these files are equivalent to the normal `/dev/crt`.

For the Series 300:

```
mknod /dev/ocrt  c  12  0x000001  for the overlay planes
mknod /dev/icrt  c  12  0x000002  for the image planes
```

For the Series 300 external configuration:

```
mknod /dev/ocrt  c  12  0x⟨sc⟩0201  for the overlay planes
mknod /dev/icrt  c  12  0x⟨sc⟩0202  for the image planes
```

For the Series 800:

```
mknod /dev/ocrt  c  14  0x00⟨lu⟩01  for the overlay planes
mknod /dev/icrt  c  14  0x00⟨lu⟩02  for the image planes
```

Linking the Driver

The HP 98550A Device Driver is located in the `/usr/lib` directory with the file name `libdd98550.a`. This device driver may be linked to a program using the absolute path name `/usr/lib/libdd98550.a` or an appropriate relative path name, or by using the `-l` option `-ldd98550`. For example: to compile and link a program for use with this driver, use:

```
cc example.c -ldd98550 -lsb1 -lsb2 -o example
fc example.f -ldd98550 -lsb1 -lsb2 -o example
pc example.p -ldd98550 -lsb1 -lsb2 -o example
```

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, e.g. `/dev/crt`.

Kind — This parameter must be `OUTDEV`, unless used for a graphics window, in which case `OUTINDEV` may be used.

Driver — The character representation of the driver type. This is `hp98548`, `hp98549`, or `hp98550` modified to meet the syntax of the programming language used, namely:

`"hp98550"`	*for C.*
`'hp98550'//char(0)`	*for FORTRAN77.*
`'hp98550'`	*for Pascal.*

The driver will correctly open any of the supported displays if any of the strings `hp98548`, `hp98549`, or `hp98550` are used. An `inquire_id`

(3g) call following the **gopen** will indicate which display was actually found.

Mode

The mode control word consists of several flag bits *or* ed together. Listed below are flag bits that have device-dependent actions. Those flags not discussed below operate as defined by the **gopen** procedure. See *Starbase Programming with X11* manual for a description of **gopen** actions when accessing an X Window.

`SPOOLED` Raster devices cannot be spooled.

`MODEL_XFORM` Shading is not supported for this device. However, opening in `MODEL_XFORM` mode will affect how matrix stack and transformation routines are performed.

`0 (zero)` Open the device, but do nothing else. The software color table is initialized from the current state of the hardware color map. The special device file's minor number determines the number of color planes used.

`INIT` Open and initialize the device as follows:

1. Frame buffer is cleared to 0s.
2. The color map is reset to its default values.
3. The display is enabled for reading and writing.
4. The overlay planes are configured with 0s transparent.

`RESET_DEVICE` Open and initialize the device as follows:

1. The hardware state is reinitialized to its boot-up state.
2. Frame buffer is cleared to 0s (all overlay and image planes).
3. The color maps are reset to default values (overlay and image).
4. The display is enabled for reading and writing.
5. The overlay planes are configured with 0s transparent.

Note SPOOLED and MODEL_XFORM flag bits have no device dependent effects.

Syntax Examples

To open and initialize an HP 98550A device for output:

C programs:

```
fildes = gopen("/dev/crt",OUTDEV,"hp98550",INIT);
```

FORTRAN77 programs:

```
fildes = gopen('/dev/crt'//char(0),OUTDEV,'hp98550'//char(0),INIT)
```

Pascal programs:

```
fildes := gopen('/dev/crt',OUTDEV,'hp98550',INIT);
```

Special Device Characteristics

Device Coordinate Addressing

For device coordinate operations, location $(0, 0)$ is the upper-left corner of the screen with X-axis values increasing to the right and Y-axis values increasing down. The lower-right corner of the display is $(1279, 1023)$ for the HP 98548A and HP 98550A displays, and $(1023, 767)$ for the HP 98549A or HP 319C display.

Offscreen Memory Usage

The HP 98550A Device Driver allocates a portion of offscreen memory each time it is opened for things like fill patterns and raster echo storage. The first 32 lines of offscreen frame buffer memory are reserved for the driver and Windows/9000 or X server. The remaining lines may be allocated for raster font bitmaps, or by the driver for raster echo storage. When the driver does allocate offscreen memory for cursors, it consumes 64 lines at a time. Storage needed by font optimization varies with the font size. The X Windows System also uses offscreen for temporary and client pixmaps.

In general, offscreen frame buffer memory is allocated by the system from top to bottom (i.e., from low offsets to high). Refer to the "Window Device Driver" chapter for further information on HP Windows/9000 use of frame buffer memory. Also review the descriptions of the `gescape` operations `R_OFFSCREEN_ALLOC` and `R_OFFSCREEN_FREE` in this manual. Use of offscreen while the X Windows Systemis running is not recommended.

Device Defaults

Dither Default

The number of colors allowed in a dither cell is 1, 2, 4, 8 or 16. The default value is 2.

Raster Echo Default

The default raster echo is the 8×8 array. Displays with less than eight color planes use the appropriate part of this pattern.

255	255	255	255	0	0	0	0
255	255	0	0	0	0	0	0
255	0	255	0	0	0	0	0
255	0	0	255	0	0	0	0
0	0	0	0	255	0	0	0
0	0	0	0	0	255	0	0
0	0	0	0	0	0	255	0
0	0	0	0	0	0	0	255

The maximum size for a raster echo is 64×64 pixels. The default drawing mode for the raster echo is 7 (a logical *OR*).

By default, all echo types are written to the open planes. The location of raster and non-raster echoes may be changed using the `gescape` operation `R_OVERLAY_ECHO`.

Plane Mask Defaults

All accessible planes display enabled. All accessible planes `write_enabled`.

Semaphore Default

Semaphore operations are enabled.

Line Type Defaults

The default line types are created with the bit patterns shown below. The Starbase default line type is SOLID, line type 0. See the following table.

Table HP98550A-4.

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

Number of Color Planes

When the gopen procedure is called, this driver identifies the device. Based on this and the desired configuration as determined by the special device minor number, the device driver then acts accordingly. Only the planes opened (1, 2, 4, 6, or 8) may be accessed by most Starbase primitives. The only exception is for cursors (see the gescape operation R_OVERLAY_ECHO).

Default Color Map

If the fourth gopen parameter is zero (0) then the current hardware color map is used on color displays.

If the fourth gopen parameter is INIT, then the current color map is initialized to the default values shown below. For the HP 98549A or HP 319C display, the color map is initialized to the first (2, 16, or 64) entries of the Starbase default color map. For the monochrome device, only the first two entries are used. For 2-overlay planes, only the first four entries are used. For 8-image planes 256 entries are used.

Overlay transparency affects the meaning of the zero entry in the overlay color map. If both overlay planes are display_enable, only the zero entry in the color map is potentially transparent. Display-disabling a plane is equivalent to zero in

that bit position. For example, if the least-significant overlay plane is disabled, only the 0 and 2 entries in the color map are active.

Table HP98550A-5. 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

Use the `inquire_color_map` procedure to see the rest of the colors.

Red, Green and Blue

Each file descriptor opened as an output device has a software color table associated with it. If multiple file descriptors are open to the same device, the software color tables and the hardware color map may not always be identical. The color table does not track the color map if the device's color map is changed via another file descriptor path.

It is usually more efficient to select a color with a color map index rather than specifying a color with red, blue and green values because of the time it takes for the driver to figure out which pen in the color table most closely matches the specified color.

Selecting a color with the `fill_color` procedure will allow dithering for filled areas when desired.

Starbase Functionality

Unsupported Procedures

The following procedures are not supported for use with this driver. Calls to these procedures will have no effect:

```
backface_control
bank_switch
bf_control
bf_fill_color
bf_interior_style
bf_perimeter_color
bf_perimeter_repeat_length
bf_perimeter_type
bf_surface_coefficients
bf_surface_model
define_trimming_curve
depth_cue
depth_cue_color
depth_cue_range
hidden_surface
interior_style (INT_OUTLINE)
interior_style (INT_POINT)
light_ambient
light_attenuation
light_model
light_source
light_switch
shade_range
surface_model
surface_coefficients
viewpoint
zbuffer_switch
```

Conditionally Supported Procedures

The following procedures are supported under the listed conditions:

`block_read`, `block_write`

The `raw` parameter for the `block_read` and `block_write` commands is used by this driver to do plane-major reads and writes. It is enabled by the gescape `R_BIT_MODE`. The storage supplied by the user as the source or destination must be organized as follows. The data from each plane will be packed, eight pixels per byte. Each row must begin on a byte boundary. The size of the rectangle as specified by the ⟨*length_x*⟩ and ⟨*length_y*⟩ parameters will thus need $((\langle length\_x \rangle + 7)/8) \times \langle length\_y \rangle$ bytes. The data from the next plane will begin on the following byte

boundary. Clipping is done to the screen limits. The first pixel in the source rectangle is placed in the high-order bit of the first byte in each plane region. If clipping is done, part of each plane region will not be read (`block_read`) or altered (`block_write`).

A bit mask selects the planes to be read or written. The initial value of this mask is 1 (one), indicating that only plane 0 is to be accessed. The value of the mask may be changed using the `R_BIT_MASK` or `GR2D_PLANE_MASK` `gescape`. `GR2D_PLANE_MASK` is discussed in the appendix of this manual. The planes selected by the mask are expected to reside in consecutive plane locations in the user storage area. This reduces the storage requirements to exactly what is needed, but also presents the potential for addressing violations or undesirable results. For example, if the plane mask is changed to specify more planes between a `block_read` and a following `block_write` from the same location, the `block_write` will attempt to access storage for planes that were not read (and perhaps not allocated). The application program must ensure consistency in these operations.

`shade_mode` The color map mode may be selected but shading cannot be turned on.

`text_precision` Only `STROKE_TEXT` precision is supported.

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

Fast Alpha and Font Manager Functionality

This device driver supports raster text calls from the fast alpha and font manager libraries. See the *Fast Alpha/Font Manager's Programmer's Manual* for further information.

Parameters for gescape

The following gescape functions are common to two or more of the Hewlett-Packard displays supported by Starbase. Detailed information about these functions can be found in Appendix A.

- BLINK_PLANES—Blink display (blink rate is 3.75 Hz for this device).
- GR2D_DEF_MASK—Defines mask.
- GR2D_FILL_PATTERN—Define 16×16 dither and fill pattern.
- GR2D_MASK_ENABLE—Enables mask rule and current mask.
- GR2D_MASK_RULE—Set three-operand drawing mode.
- GR2D_OVERLAY_TRANSPARENT—Turns on/off transparency of 0 pixels.
- GR2D_PLANE_MASK—Overrides the mask.
- GR2D_REPLICATE—Allows square pixel replication.
- R_BIT_MASK—Bit mask.
- R_BIT_MODE—Bit mode.
- R_DEF_ECHO_TRANS—Turns on transparency.
- R_DEF_FILL_PAT—Define fill pattern.
- R_GET_FRAME_BUFFER—Read frame buffer address.
- R_LOCK_DEVICE—Lock device.
- R_OFFSCREEN_ALLOC—Allocates offscreen frame buffer memory.
- R_OFFSCREEN_FREE—Frees allocated offscreen frame buffer memory.
- R_OVERLAY_ECHO—Allows cursor in overlay or graphics planes.

Note Using R_OVERLAY_ECHO to move the cursor into the overlay planes will interfere with graphics done directly to those planes. Overlay cursors are not supported in all window system configurations.

- R_UNLOCK_DEVICE—unlock device
- READ_COLOR_MAP—read color map
- SWITCH_SEMAPHORE—semaphore control

Performance Tips

- A solid color for polygon fill (set by `fill_color_index`), in either two-operand or three-operand mode, will fill faster than a dither or fill pattern (set by `fill_color` or one of the fill-pattern `gescape` operations).
- Certain two-operand drawing modes may be done faster than others. The absolute modes (`ZERO` (0) and `ONE` (15)) are the fastest. Rules dependent only on one operand (e.g., ⟨*source*⟩ (3) or `not` ⟨*dest*⟩ (10)) are somewhat slower. Rules dependent on both operands (e.g., `xor` (6)) are the slowest.
- Placing the Starbase echoes in the overlay planes may improve performance because the driver does not have to "pick up" the cursor to draw to the image planes.
- Buffering of graphics operations is done in this driver to enhance performance. If `buffer_mode` is turned off or many calls are made to `make_picture_current` then performance may decrease.
- Performance optimizations have been made so that sequential calls of the same output primitive with no intervening attribute change or call to a different primitive are processed faster. For example, the sequence `polygon, polygon, polyline, polyline` is faster than `polygon, polyline, polygon, polyline`. The `line_color, polyline, polyline` calls are faster than `line_color, polyline, line_color, polyline`.

- For the best performance when using bit/pixel block write (`raw` mode `TRUE`, `R_BIT_MODE` enabled), the following conditions must be met:

    1. Source rows should be an even number of whole bytes (that is, `dx` should be a multiple of 16).
    2. Destination rows should be aligned on 8-pixel boundaries (that is, `x` should be a multiple of 8).
    3. Source rows should be aligned.

Note When drawing in a graphics window, drawing and filling performance will be significantly lower if the window raster extends more than 1024 device coordinates outside the screen in any direction, either because of its size or its current position on the screen. There is a significant additional performance cost associated with drawing to a retained rather than an unretained raster.

Contents

HP98556

The HP 98556 Device Driver

Device Description

The HP 98556 Device Driver is used to interface the Starbase Graphics Library with the HP 98556 Graphics Accelerator Board. The HP 98556 Graphics Accelerator is an optional printed circuit board which can mate with either the HP 98549A 1024×768 Display Board, or the HP 98550A 1280×1024 Display Board. The accelerator board is plugged into the display board, which is then inserted in a system slot. The accelerator is supported on the HP 9000 Series 300 workstations and the Series 800 workstations (see table 1-8 in the Introduction section of this manual). This configuration allows for use of multiple high speed windows on HP 9000, HP Windows/9000 and the X Window system.

The HP 98556 driver should be used when speed is very important and integer or dc graphics operations are performed. When speed in graphics operation performance is not as important, the HP 98550 driver should be used. The HP 98556 driver only supports 31 simultaneous **gopens**. Any additional **gopens** must be to the HP 98550 driver.

The HP 98556 Graphics Accelerator has hardware and micro-code support for the functions of the HP 98549A and HP 98550A workstations, plus fast two-dimensional and integer transformations, clipping, and primitive drawing. The HP 98556 Device Driver allows use of the features of the HP 98556 accelerator.

Hardware Overview

Frame Buffers

The display supported by this driver possesses both **image** and **overlay** planes. A major difference between image and overlay planes is depth. The HP 98549A board supports either four image planes and two overlay planes or six image planes. The HP 98550A board supports eight image planes and two overlay planes. The two overlay planes index their own four-entry color map, while the image planes index the main 256-entry color map. The overlay planes can be thought of as being "on top of" the image planes.

The four colors provided by the overlay planes are denoted as colors 0, 1, 2, and 3. Colors 1, 2, and 3 in the overlay planes are always dominant and displayed for the corresponding pixels, regardless of what is in the image planes. Color 0, however, may be made transparent, allowing the image planes to show through.

For example, a cursor might be drawn in the overlay planes with color 1. If zeroes are found everywhere else in the overlay planes and they are transparent, the cursor will be dominant but independent of the image. This speeds up drawing because a cursor drawn directly in the image planes (by complementing the image for example) must normally be removed before drawing. The driver may avoid these steps if it knows the cursor is not in the image planes.

The `gescape GR2D_OVERLAY_TRANSPARENT` may be used to make overlay color 0 dominant, i.e., all four overlay plane colors are displayed and the image planes are entirely obscured. The default mode, color 0, is transparent.

Typically, the user does not need direct access to pixels in the frame buffer. However, for applications that require direct access, Starbase provides the `gescape R_GET_FRAME_BUFFER` that returns the virtual memory address of the beginning of the frame buffer (this `gescape` is discussed later in more detail). Frame buffer locations are then addressed relative to the returned address, in byte-per-pixel mode.

Series 800 When writing to IO space, accesses must be on word (32 bit) boundaries. The frame buffer is mapped as one word per pixel; therefore, pixels should be addressed on word boundaries when directly accessing the frame buffer. Also, the pixel value is in the least significant byte of the word.

To ensure valid direct frame buffer access, the user must precede the `R_GET_FRAME_BUFFER gescape` with the `R_LOCK_DEVICE gescape`. After completing the frame buffer access and prior to any other Starbase commands, the user must call the `R_UNLOCK_DEVICE gescape`.

The HP 98549A Display

The HP 98549A Display System has six image planes (or four image plus two overlay planes, definable through the minor number in the device file). Resolution is 1024×768 pixels.

The frame buffer is 1024×1024 bytes. The bottom 256 lines of the frame buffer are not displayed and are used for temporary storage of graphical items. This off-screen portion of the frame buffer may be accessed via the `gescape R_FULL_FRAME_BUFFER` documented later. Care should be taken when using this `gescape` since other processes, Starbase, and the window system also access the frame buffer off-screen memory.

The first byte (byte 0) of the frame buffer represents the upper left corner pixel of the screen. Byte 1 is immediately to its right. Byte 1023 is the last (right-most) pixel on the top line. Byte 1024 is the first (left-most) pixel on the second line from the top. The last (lower right corner) pixel on the screen is byte number 786,431 (767×1024+1023).

Creation of the special device files used to access the device in these modes follows. Selection of the mode is done when the file is opened with the `gopen` procedure.

Note Do not open the HP 98549A simultaneously in both 6-plane mode and 4+2-plane mode. Doing so will cause indeterminate results. You may simultaneously open the overlay planes and the 4-image planes using two different file descriptors. If graphics are being done in the overlay planes, do not use the gescape

R_OVERLAY_ECHO to move the cursor from the image planes into the overlay planes becaues this will interfere with the overlay graphics. Only the 6-plane mode is supported by the X windows system.

For normal (non-raw) block read and write operations, the data is in the least significant bits of each byte. The number of valid bits depends on the logical device opened (2, 4, or 6 valid bits).

The gescape R_GET_FRAME_BUFFER returns the virtual memory address of the beginning of the frame buffer. Caution is necessary if the HP 98549 is opened in the 4+2-plane mode, because the frame buffer address returned for the 4-image planes is the same as the address for the 2-overlay planes. To ensure that you only access the planes opened, the R_LOCK_DEVICE gescape (using the file descriptor for the appropriate planes) should be used to lock the device before reading or changing the frame buffer. Note that your program must shift each data byte to the left by four bits in order to write it to the overlay planes. Use the R_LOCK_DEVICE gescape to unlock the device after the access.

The HP 98550A Display

The HP 98550A display has eight image planes, and two overlay planes. Resolution is 1280×1024 pixels.

The frame buffer is 2048×1024 bytes. The right most 768 columns of the frame buffer are not displayed but used for temporary storage of graphical items. This off-screen portion of the frame buffer may be accessed via the gescape R_FULL_FRAME_BUFFER documented later. Care should be taken when using this gescape since other processes, the Starbase driver, and the window system also access the frame buffer off-screen memory.

The first byte (byte 0) of the frame buffer represents the upper left corner pixel of the screen. Byte 1 is immediately to its right. Byte 1279 is the last (right-most) pixel on the top line. The next 768 bytes are not displayable. Byte 2048 is the first (left-most) pixel on the second line from the top. The last (lower-right corner) pixel on the screen is byte number 2,096,383 (1023×2048+1279).

Color Map

The frame buffer of the display system is organized as a set of planes. For the HP 98549A Display Board, the six (or four) LSBs of each byte determine the color, providing color values from 0–63 (or 0–15). For the HP 98550A display, each byte (8 bits) is used to provide color values from 0–255. These color values are used to address the color map. The color map is a RAM table that has 16, 64, or 256 addressable locations and is 24 bits wide (8 bits each for red, green, blue). Thus, the pixel value in the frame buffer indexes the color map, generating the color programmed at that location.

Windows on the HP 98556

Note Windows/9000 is supported on the Series 300 computers only.

The HP 98556 driver supports hardware accelerated windows. Clip information for each window that is gopened with the HP 98556 driver is downloaded to the HP 98556 accelerator by the window manager or server. This enables the HP 98556 accelerator to clip output to the visible parts of the window. The clip information for each window consists of a list of visible rectangles. Performance of output to an obscured window will degrade linearly as a function of the number of visible rectangles.

The HP 98556 driver imposes a limit of 32 simultaneous gopen's of the HP 98556 device. This limit also applies to the window system. The window manager will gopen the HP 98556 device, allowing up to 31 windows to use the HP 98556 driver. Once a window is gopened with the HP 98556 driver, it counts against this 31 window limit until the window is closed.

If an open of the HP 98556A is performed when 31 open commands of the HP 98556A are currently active, a Starbase error is generated and the open command will fail. When one of the previous HP 98556A opens is closed, the first open command can be tried again.

The HP Windows/9000 window manager will default the `WMDRIVER` environment variable to the HP 98556 driver if the HP 98556 is installed. If you want it to use the HP 98550 driver instead, set `WMDRIVER` to HP 98550.

When the window manager or X server uses the HP 98556 driver, graphics windows can be gopened with both the HP 98550 and HP 98556 drivers. Windows that are gopened with the HP 98550 driver are not counted against the limit of 31 accelerated windows. (The terminal emulator windows use only the HP 98550 driver to make all 31 accelerated windows available to your programs.) Note that the HP 98556 driver cannot be used to open a window when the HP Windows/9000 window manager has the `WMDRIVER` environment variable set to HP 98550.

When a graphics image is drawn to the obscured portion of the window, only the visible parts of the window are drawn to; the obscured parts are ignored. An application should monitor the `SIGWINDOW` signal (see the HP *Windows/9000 Programmer's Manual* and `signal`(2)) and repaint the entire image when previously obscured parts of the window become visible. In the X Windows System, the application should handle exposure events.

The HP 98556 driver does not support retained windows. Output to obscured parts of a retained window will not affect the retained raster. In order to be compatible with older applications that require retained rasters, the HP 98556 driver behaves as follows when it is used to gopen a retained window (HP Windows/9000 only):

1. If the HP 98550 driver is also linked into the user program, Starbase will substitute the HP 98550 driver for the HP 98556 driver during gopen. A Starbase warning of "`Driver name substituted on gopen`" will be generated during the gopen.
2. If the HP 98550 driver is not linked into the user program, Starbase will use the HP 98556 driver. A Starbase warning of "`Driver doesn't support retained rasters`" will be generated during the gopen. The gopen will succeed, but remember that output to the obscured parts of a window will not be saved in the retained raster.

The Windows/9000 system can be opened only in the image planes. Windows in the image planes behave in the usual way. Of course, the overlay planes must be transparent to see windows (or any data) in the image planes.

Starbase Echo Operation

Only one Starbase echo is supported in a window by the HP 98556 driver. When a window is opened multiple times by the HP 98556 driver, only one of these opens

should specify a Starbase echo because the HP 98556 driver can "pick up" only one Starbase echo and one X11 cursor. When a window is opened twice by the HP 98556 driver and each open specifies a Starbase echo, the first invocation of the driver will not be able to pick up the echo generated by the second invocation of the driver.

Setting Up the Device

The HP 98556 Device Driver can only be used if the display is configured in external address space. The HP 98556 accelerator card will plug into an HP 98549A or HP 98550A display controller board.

Switch Settings

No switches on the HP 98556 board need to be set. For the switch settings of the HP 98549A/HP 98550A board, see the "HP 98550 Device Driver" chapter.

Special Device Files (mknod)

The `mknod` command (see `mknod(8)` man page), creates a special device file that is used to communicate between the computer and the display device. 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.

The Series 300 `mknod` parameters are character device (`c`) with a major number of `12` and a minor number of `0x`⟨*sc*⟩`020`⟨*d*⟩, where ⟨*sc*⟩ is a two digit select code and ⟨*d*⟩ is a single digit denoting which planes should be opened by the driver. The Series 800 `mknod` parameters are character device (`c`) with a major number of `14` and a minor number of `0x00`⟨*lu*⟩`0`⟨*d*⟩ where ⟨*lu*⟩ is the logical unit number of the A1020A graphics subsystem, and ⟨*d*⟩ is a single digit denoting which planes should be open by the driver. 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 `crt`. Note, the leading `0x` causes the number to be interpreted hexadecimally.

The normal device file (the last digit at the minor number is zero) will open the driver to the image planes (creating a special device file that uses all six of the planes for the HP 98549A device as image planes).

Series 300

```
mknod /dev/crt  c  12  0x⟨sc⟩0200
```

Series 800

```
mknod /dev/crt  c  14 0x00⟨lu⟩00
```

To open the driver for the 2-overlay planes only, the last digit of the minor number must be 1. Similarly, to access only the image planes (a HP 98549A 4+2 configuration or HP 98550A), the last digit of the minor number must be 2.

Series 300

```
mknod /dev/ocrt   c   12   0x⟨sc⟩0201  (for the overlay planes)
mknod /dev/icrt   c   12   0x⟨sc⟩0202  (for the image planes)
```

Series 800

```
mknod /dev/ocrt   c   14   0x00⟨lu⟩01  (for the overlay planes)
mknod /dev/icrt   c   14   0x00⟨lu⟩02  (for the image planes)
```

You must be superuser or root to use the `mknod` command.

Linking the Driver

The HP 98556 Device Driver is located in the `/usr/lib` directory with the file name `libdd98556.a`. This device driver may be linked to a program using the absolute path name `/usr/lib/libdd98556.a`, an appropriate relative path name, or the `-l` option `-ldd98556`. For example: to compile and link a program for use with this driver, use:

```
cc example.c -ldd98556 -lsb1 -lsb2 -o example
fc example.f -ldd98556 -lsb1 -lsb2 -o example
pc example.p -ldd98556 -lsb1 -lsb2 -o example
```

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/crt`.

`Kind` — Indicates the I/O characteristics of the device. This parameter must be `OUTDEV`, unless used for a graphics window, in which case `OUTINDEV` may be used.

Driver — The character representation of the driver type. This is `hp98556` modified to meet the syntax of the programming language used, namely:

`"hp98556"`	*for C.*
`'hp98556'//char(0)`	*for FORTRAN77.*
`'hp98556'`	*for Pascal.*

Note

In HP Windows/9000, the HP 98550 driver will be used if an attempt is made to open an HP 98556 driver to a retained window. Retained rasters are not supported by the HP 98556 driver.

Mode — The mode control word consists of several flag bits that are *or*ed together. Listed below are those flag bits that have device-dependent actions. Those flags not discussed below operate as defined by the gopen procedure. (See *Starbase Programming with X11* manual for a description of the actions of gopen in an X window.)

- `SPOOLED`—Cannot spool to raster devices.
- `0`—Open the device, but do nothing else. The software color map is initialized from the current hardware color map. The special device file's minor number specifies the number of image planes that are used.
- `INIT`—Open and initialize the device as follows:
    1. Image planes and/or overlay planes are cleared to 0s.

2. The color map is reset to its default values, if using the image plane configuration.
3. The display is enabled for reading and writing.
4. The overlay planes are configured such that 0's are transparent.

- RESET_DEVICE—Open and initialize the device as follows:
  1. The hardware state is reinitialized to its boot-up state.
  2. Frame buffer is cleared to 0's (all image and overlay planes).
  3. The color maps are reset to their default values (overlay and image.)
  4. The display is enabled for reading and writing.
  5. The overlay planes are configured such that 0's are transparent.
  6. Download the transform engine's microcode.
- INT_XFORM—Open and initialize the device to allow for fast integer transformations.
- MODEL_XFORM—Shading is not supported for this device. The MODEL_XFORM matrix stack definition is supported.

Syntax Examples

To open and initialize an HP 98556 device for output:

C Syntax Example

```
fildes = gopen("/dev/crt",OUTDEV,"hp98556",INIT);
```

FORTRAN77 Syntax Example

```
fildes = gopen('/dev/crt'//char(0), OUTDEV,'hp98556'//char(0),INIT)
```

Pascal Syntax Example

```
fildes := gopen('/dev/crt',OUTDEV,'hp98556',INIT);
```

Offscreen Memory Usage

On the Series 300 computers, each time the HP 98556 Device Driver is opened, it allocates a portion of offscreen memory. This memory is used for such things as raster echo storage. The allocation and storage is as follows.

The first 32 lines of offscreen frame buffer memory are reserved for the Windows/9000 sprite. The remaining lines may be allocated by Windows/9000 for raster font bitmaps or by the driver for raster echo storage. When the driver allocates offscreen memory, it consumes 64 lines at a time. Storage required by font optimization varies with the font size.

In general, offscreen frame buffer memory is allocated by the system from top to bottom (i.e., from low offsets to high). Please refer to the "Window Device Driver" chapter in the *Starbase Device Drivers Library* for further information on HP Windows/9000 use of frame buffer memory. Also, see the descriptions of the `gescapes` `R_OFFSCREEN_ALLOC` and `R_OFFSCREEN_FREE`.

Special Device Characteristics

For device coordinate operations, location (0, 0) is the upper-left corner of the screen with X-axis values increasing to the right and Y-axis values increasing down. The lower-right corner of the HP 98549A display is (1023,767) while the lower-right corner of the HP 98550A display is (1279,1023).

Device Defaults

Number of Planes

For the HP 98549A display, there will be either four image plus two overlay planes, or six image planes (definable by the special device file). For the HP 98550A display there are eight image plus two overlay planes.

Dither Default

The default number of colors searched for in a dither cell is 2. The number of colors allowed in a dither cell is 1, 2, 4, 8 or 16.

Raster Echo Default

The default raster echo is the 8×8 array (displays with less than eight image planes use the appropriate number of least significant bits of this pattern):

```
255 255 255 255   0   0   0   0
255 255   0   0   0   0   0   0
255   0 255   0   0   0   0   0
255   0   0 255   0   0   0   0
  0   0   0   0 255   0   0   0
  0   0   0   0   0 255   0   0
  0   0   0   0   0   0 255   0
  0   0   0   0   0   0   0 255
```

The maximum size allowed for a raster echo is 64×64 pixels. The default drawing mode for the raster echo is 7 (a logical *OR*). By default, all echo types are written to the open planes. The location of raster and non-raster echoes may be changed by using the `R_OVERLAY_ECHO gescape`.

Plane Defaults

All planes being used are display enabled and write enabled.

Semaphore Default

Semaphore operations are enabled.

Line Type Defaults

The default line types are created with the bit patterns shown below. The Starbase default line type is `SOLID`, line type `0`.

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

Default Color Map

If the fourth **gopen** parameter is zero (0), the current hardware color map is used on color displays.

If the fourth **gopen** parameter is INIT, the current color map is initialized to the default values shown below. For two overlay planes, the first 4 entries are used to initialize their independent color map (that is, 1 = white, 2 = red, 3 = yellow, and 0 defaults to transparent). For four image planes, 16 entries are used; for six image planes, 64 entries are used; and for eight image planes, 256 entries are used.

Overlay transparency affects the meaning of the zero entry in the overlay color map. If both overlay planes are display-enabled, only the zero entry of the color map is potentially transparent. Display-disabling a plane is equivalent to zero in that bit position. For example, if the least-significant overlay plane is disabled, only the 0 and 2 entries in the color map are active.

A **gescape** (GR2D_OVERLAY_TRANSPARENT) can be used to make overlay color 0 dominant. Then, all four overlay plane colors are displayed and the image planes are entirely obscured. The default mode, color 0, is transparent.

Table HP98556-1. 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

Use the `inquire_color_table` procedure to see the rest of the available colors.

Red, Green and Blue

Each file descriptor opened as an output device has a software color table associated with it. If multiple file descriptors are open to the same device, the software color table and the hardware color map may not always be identical. The color table does not track the color map if the device's color map is changed by another file descriptor path.

It is usually more efficient to select a color with a color map index rather than specifying a color with red, green and blue values. It takes longer for the driver to figure out which pen in the color table most closely matches the specified color.

Starbase Functionality

Commands Not Supported

These procedures are ignored.

```
backface_control              hidden_surface
bank_switch                   interior_style (INT_OUTLINE)
bf_control                    interior_style (INT_POINT)
bf_fill_color                 light_ambient
bf_interior_style             light_attenuation
bf_perimeter_color            light_model
bf_perimeter_repeat_length    light_source
bf_perimeter_type             light_switch
bf_surface_coefficients       shade_range
bf_surface_model              surface_model
define_trimming_curve         surface_coefficients
depth_cue                     viewpoint
depth_cue_color               zbuffer_switch
depth_cue_range
```

Commands Conditionally Supported

The following commands are supported under the listed conditions:

`block_read`, `block_write`

The `raw` parameter for the `block_read` and `block_write` commands is used by this driver to perform plane-major reads and writes. It is enabled by the `gescape R_BIT_MODE`. The storage supplied by the user as the source or destination must be organized as follows. The data from each plane will be packed, 8-pixels per byte. Each row must begin on a byte boundary. The size of the rectangle as specified by the ⟨*length_x*⟩ and ⟨*length_y*⟩ parameters will thus require $((\langle length\_x \rangle + 7)/8) \times \langle length\_y \rangle$ bytes. The data from the next plane will begin on the following byte boundary. Clipping is done to the screen limits. The first pixel in the source rectangle is placed in the high-order bit of the first byte in each plane region. If the plane region gets clipped, part of each

plane region will not be read (**block_read**) or altered (**block_write**).

A bit mask selects the planes to be read or written. The initial value of this mask is 1 (one), indicating that only plane 0 is to be accessed. The value of the mask may be changed using the **R_BIT_MASK** or **GR2D_PLANE_MASK** **gescape**. **GR2D_PLANE_MASK** is discussed later in this chapter. The planes selected by the mask are expected to reside in *consecutive* plane locations in the user storage area. This reduces the storage requirements to exactly what is needed but also presents the potential for addressing violations or undesirable results. For example, if the plane mask is changed to specify more planes between a **block_read** and a following **block_write** from the same location, the **block_write** will attempt to access storage for planes that were not read (and perhaps not allocated). The application program must ensure consistency in these operations.

inquire_color_table — When running in the overlay planes, this command returns the software color map values.

shade_mode — The color map mode may be selected, but shading cannot be turned on.

text_precision — Only **STROKE_TEXT** precision is supported.

vertex_format — The ⟨*use*⟩ parameter must be zero, and extra coordinates supplied will be ignored.

Fast Alpha and Font Manager Functionality

This device driver supports raster text calls from the fast alpha and font manager libraries. See the *Fast Alpha/Font Manager's Programmer's Manual* for further information.

Parameters for gescape

The following gescape functions are common to two or more of the Hewlett-Packard displays supported by Starbase. Detailed information about these functions can be found in Appendix A.

- BLINK_PLANES—Blink display (blink rate is 3.75 Hz for this device).
- GR2D_DEF_MASK—Defines mask.
- GR2D_FILL_PATTERN—Define 16×16 dither and fill pattern.
- GR2D_MASK ENABLE—Enables mask rule and current mask.
- GR2D_MASK_RULE—Set three operand drawing mode.
- GR2D_OVERLAY_TRANSPARENT—Turns on/off transparency of 0 pixels.
- GR2D_PLANE_MASK—Overrides the mask.
- GR2D_REPLICATE—Allows square pixel replication.
- R_BIT_MASK—Bit mask.
- R_BIT_MODE—Bit mode.
- R_DEF_FILL_PAT—Defines the current 4×4 pixel dither cell.
- R_FULL_FRAME_BUFFER—Map in offscreen.
- R_GET_FRAME_BUFFER—Read frame buffer address.
- R_LOCK_DEVICE—Lock device.
- R_OFFSCREEN_ALLOC—Allocates offscreen frame buffer memory.
- R_OFFSCREEN_FREE—Frees allocated offscreen frame buffer memory.
- R_OVERLAY_ECHO—Select plane to contain cursor.

- R_UNLOCK_DEVICE—Unlock device.
- READ_COLOR_MAP—Read color map.
- SWITCH_SEMAPHORE—Semaphore control.

The following **gescape** function is unique to this driver. It is discussed in the next section.

- GR2D_CONVEX_POLYGONS—Enables convex polygons to be drawn at a higher speed.

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

```
/* 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

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

arg1(1)=TRUE;
call gescape(fildes,GR2D_CONVEX_POLYGONS,arg1,arg2)
```

Pascal Syntax

```
{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);
```

Performance Tips

1. A solid color for polygon fill (set by `fill_color_index`), in either two-operand or three-operand mode, will fill faster than a dither or fill pattern (set by `fill_color` or `pattern_define`).
2. Certain two-operand drawing modes may be performed faster than others. The absolute modes (`ZERO` (0) and `ONE` (15)) are the fastest. Rules dependent on only one operand (e.g., ⟨*source*⟩ (3) or `not` ⟨*dest*⟩ (10)) are a bit slower. Rules dependent on both operands (e.g., `xor` (6)) are the slowest.
3. Placing Starbase echoes in the overlay planes will improve performance because the driver does not have to "pick up" the cursor to draw to the image planes.
4. Buffering of graphics operations is done in this driver to enhance performance. If `buffer_mode` is turned off or many calls are made to `make_picture_current`, performance may decrease.
5. Performance optimizations have been made so that sequential calls of the same output primitive with no intervening attribute change, or call to a different primitive, are processed faster. For example, the sequence `polygon, polygon, polyline, polyline` is faster than `polygon, polyline, polygon, polyline`. In a similar way, `line_color, polyline, polyline` is faster than `line_color, polyline, line_color, polyline`.
6. Buffer polylines.

   Each Starbase call requires a certain amount of overhead. If a number of primitives can be bundled into a single Starbase call (like a polyline, for instance), the amount of system overhead required per primitve will be significantly reduced.

Therefore, you should try to convert individual move/draws into polylines. Likewise, the larger you can make a polyline (that is, the more vertices it contains) the faster the system will draw it.

For example, the following:

```
move
draw
draw
rectangle
draw
move
draw
circle
move
draw
```

could be done faster as:

```
polyline (containing the moves/draws)
rectangle
circle
```

7. Convert primitives to polylines.

   Polylines are the fastest primitive. By converting other primitives to polylines, you will be able to draw them at the polyline rate. Examples of primitives that could be converted to polylines would be; edged non-filled rectangles, edged non-filled polygons, etc. An exception to this is polycircles, which are always faster than constructing circles out of polylines (polycircles are only available for the 6.5 and later releases).

8. Minimize attribute switching.

   Setting up the system for new primitive attributes (fill colors, line colors, fill styles, line types, etc.) is a fairly expensive operation, timewise. Therefore, avoid redundant attribute switching. Try to group primitives with similar attributes together.

Take for example the following sample calls:

```
fill_color_index(white)
rectangle A
fill_color_index(red)
rectangle B
fill_color_index(white)
rectangle C
fill_color_index(red)
rectangle D
fill_color_index(white)
rectangle E
```

This could be done much more effeciently as:

```
fill_color_index(white)
rectangle A
rectangle C
rectangle E
fill_color_index(red)
rectangle B
rectangle D
```

9. Limit the use of polygons.

   Polygons are a relatively slow primitive. Whenever possible, try to convert hollow polygons to polylines, or failing that, use rectangles.

10. Use convex polygons as opposed to concave polygons whenever possible.

    Convex polygons will be drawn faster than concave polygons, so use them whenever possible. This is especially true if the `GR2D_CONVEX_POLYGONS` `gescape` available in the Series 300 6.5 release, Series 800 3.1 release and later releases is used. This `gescape` improves performance by taking advantage of optimiaztions made in the HP 98556 device firmware.

11. Make use of polycircle and polyrectangle commands. (6.5 release and later only)

    Whenever possible, use the polycircle and polyrectangle commands. Remember however, that many consecutive, edged, non-filled rectangles will still be drawn faster as polylines. Use the largest polycircles and polyrectangles (that is, greatest number of primitives) practical for your application.

Here again, the larger you make the polyrectangle or polycircle, the less time per primitve is spent on system overhead.

12. Group rectangles, circles, and text together.

    From the Series 300 6.5 and Series 800 3.1 releases and on, the HP 98556 device is designed to have a significant performance advantage in grouping rectangles with other rectangles and text with other text. Also, try to group circles of the same radius together. Failing that, group circles of different radii together. To take full advantage of this feature, do not change attributes (that is. `fill_color`, `interior_style`, etc.) between the primitives. The most dramatic performance improvements from this technique are seen in 6.5 and later releases; however, some improvement will be seen in earlier releases, too.

13. Use a display list whenever possible.

    Your overall system performance will not be any faster than the slowest element in the pipeline. For many applications, the HP 98556 system is capable of processing primitives significantly faster than the application can send them. To alleviate this situation, the use of a display list (such as Hewlett-Packard's Starbase Display list) is recommended.

Cautions

The Internal Terminal Emulator (ITE) operates in the image planes only, not the overlay planes. There can be interactions if graphics and the ITE are active simultaneously in the image planes.

The HP 98556 hardware only supports 31 accelerated simultaneous gopens across all processes. If more than 31 gopens are desired, those that need not be accelerated should use the HP 98550 Device Driver.

Polygons of up to 255 vertices are supported. If a polygon has more than 255 vertices, only the first 255 vertices are displayed.

Contents

HP98700

The HP 98700 Device Driver

Device Description

Two device drivers are provided to access the HP 98700 display:

- HP 98700—used to access the graphics display without using the optional graphics accelerator.
- HP 98710—used to access the graphics display using only the optional graphics accelerator.

This section covers the HP 98700 Device Driver; see the "HP 98710 Device Driver" section for information on the HP 98700 with the optional graphics accelerator.

The HP 98700H Graphics Display Station includes a high-resolution 19-inch color display, a display controller, an optional keyboard, and an optional graphics accelerator (see "HP 98710 Device Driver" section). An interface is provided for this device which plugs into an I/O slot on the Series 300 SPUs.

The display has a resolution of 1024×768 pixels. The color display system comes standard with four planes of frame buffer to provide 16 simultaneous colors. An optional four additional planes of frame buffer may be installed providing 256 simultaneous colors. A color map provides 8 bits per color (for red, green and blue), providing a color palette of over 16 million colors.

The display system is a bit-mapped device with special hardware for:

- write-enabling planes
- displaying planes
- writing pixels to the frame buffer with a given replacement rule (see `drawing_mode`)
- blinking planes
- moving a block of pixels from one place in the frame buffer to another on 4-pixel boundaries

The display is organized as an array of bytes, with each byte representing a pixel on the display. With four planes installed, the four Least Significant Bits (LSBs) of each byte determine the color, providing color values from 0–15. When eight planes are installed, color values range from 0–255. These values are used to address the color map. The color map is basically a RAM table that has 16 or 256 addressable locations and is 24 bits wide (8 bits each for red, green and blue). Thus, the pixel value in the frame buffer addresses the color map, generating the color programmed at that location.

Typically, the user does not need to directly read or write pixels in the frame buffer. However, for those applications which require direct access, Starbase does provide the `gescape` function `R_GET_FRAME_BUFFER`, which returns the virtual memory address of the beginning of the frame buffer. This `gescape` is discussed in the appendix. Frame buffer locations are then addressed relative to the returned address. The first byte of the frame buffer (byte 0) represents the upper-left corner pixel of the screen. Byte 1 is immediately to its right. Byte 1023 is the last (right-most) pixel on the top line. Byte 1024 is the first (left-most) pixel on the second line from the top. The last (lower-right corner) pixel on the screen is byte number 786,431 (767×1024+1023).

The actual frame buffer is 1024×1024 bytes. The last 256 lines of the frame buffer are not displayed and are used for temporary storage of graphical items. This off-screen portion of the frame buffer may be accessed via the `gescape` function `R_FULL_FRAME_BUFFER`, also documented in the appendix. Care should be taken when using this `gescape` since other processes (for example, Starbase and the window system) access the frame buffer off-screen memory. Starbase uses the first and last line of off-screen memory.

After reading this section, refer to the section "Windows/9000 Device Driver" to find out how this device driver can be used with Windows/9000.

Setting Up the Device

Switch Settings

On the Series 300, the HP 98700 may be configured as an external display. This is done by setting switch two (SW-2) on the HP 98287A interface card. SW-2 is the eight-switch group. Place the interface card on an anti-static surface with the external connector and dust cover plate *away from you* and the bus connection pads *toward you.* This orientation is the same as the example shown below. If the third switch from the left is set to 1 (set toward you), then the next five switches give the binary value of the HP 98700's select code. If the third switch is set to 0, the HP 98700 is an internal device (the default condition). For example, the following figure shows the switch settings necessary for the HP 98700 to be used as an external device, at select code 25.

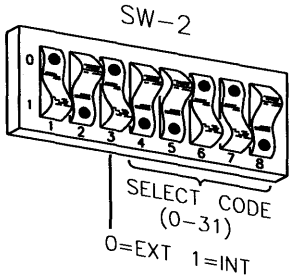


Figure HP98700-1. HP 98287A Switch Setting

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 details. 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 `crt`.

The following example 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

When the device is at an internal address (see "Switch Settings"), the `mknod` parameters are: Character device with a major number of 12 and a minor number of 0.

```
mknod /dev/crt c 12 0x000000
```

When the device is at an external address (see "Switch Settings"), the `mknod` parameters are: Character device with a major number of 12 and a minor number of `0x`⟨*sc*⟩`0200` where ⟨*sc*⟩ is the two-digit external select code. Note that the leading `0x` causes the number to be interpreted hexadecimally.

```
mknod /dev/crt c 12 0x⟨sc⟩0200
```

Linking the Driver

The HP 98700 Device Driver is located in the `/usr/lib` directory with the file name `libdd98700.a`. This device driver may be linked to a program using the absolute path name `/usr/lib/libdd98700.a` or an appropriate relative path name, or by using the `-l` option `-ldd98700`. For example, to compile and link a program for use with this driver, use:

```
cc example.c -ldd98700 -lsb1 -lsb2 -o example
fc example.f -ldd98700 -lsb1 -lsb2 -o example
pc example.p -ldd98700 -lsb1 -lsb2 -o example
```

depending upon the language being used.

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, e.g., `/dev/crt`.

Kind — This indicates the I/O characteristics of the device. This parameter must be `OUTDEV` for this driver.

Driver — This is the character representation of the driver type. This is `hp98700` modified to meet the syntax of the programming language used, namely:

`"hp98700"`	*for C*
`'hp98700'//char(0)`	*for FORTRAN77*
`'hp98700'`	*for Pascal*

Mode — This is the mode control word, which consists of several flag bits which are *or* ed together. Listed below are those flag bits which have no affect for this driver and those which have device-dependent actions. Those flags not discussed below operate as defined by the gopen procedure.

SPOOLED	Cannot spool raster devices.
0	Open the device, but do nothing else. The software color map is initialized on monochrome monitors.
INIT	Open and initialize the device as follows: 1. Frame buffer is cleared to 0s. 2. The color map is reset to its default values. 3. The display is enabled for reading and writing.

Including SPOOLED in the mode parameter has no affect on this driver. Including 0 or INIT causes a device-dependent action.

Syntax Examples

To open and initialize an HP 98700 device for output:

For C Programs:

```
fildes = gopen("/dev/crt",OUTDEV,"hp98700",INIT);
```

For FORTRAN77 Programs:

```
fildes = gopen('/dev/crt'//char(0), OUTDEV,'hp98700'//char(0),INIT)
```

For Pascal Programs:

```
fildes = gopen('/dev/crt',OUTDEV,'hp98700',INIT);
```

Special Device Characteristics

For Device Coordinate operations, location $(0,0)$ is the upper-left corner of the screen with X-axis values increasing to the right and Y-axis values increasing down. The lower-right corner of the display is therefore $(1023,767)$.

Device Defaults

Number of Color Planes

When the gopen procedure is called, this driver asks the device for the number of color planes available. This number can be either four or eight. The device driver then acts accordingly.

Dither Default

The default number of colors searched for in a dither cell is 2. The number of colors allowed in a dither cell is 1, 2, 4, 8 or 16.

Raster Echo Default

The default raster echo is the 8 by 8 array:

255	255	255	255	0	0	0	0
255	255	0	0	0	0	0	0
255	0	255	0	0	0	0	0
255	0	0	255	0	0	0	0
0	0	0	0	255	0	0	0
0	0	0	0	0	255	0	0
0	0	0	0	0	0	255	0
0	0	0	0	0	0	0	255

The maximum size allowed for a raster echo is 64×64 pixels. The default drawing mode for the raster echo is 7 (a logical *OR*).

Color Planes Defaults

All planes are display-enabled; all planes are write-enabled.

Semaphore Default

Semaphore operations are enabled.

Line Type Defaults

The default line types are created with the bit patterns in the following table. The Starbase default line type is `SOLID`, line type `0`.

Table HP98700-1.

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

Default Color Map

If the fourth **gopen** parameter is zero (0), the current hardware color map is used on color displays.

If the fourth **gopen** parameter is `INIT`, the current color map is initialized to show the following default values.

Table HP98700-2. 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

Use the `inquire_color_table` procedure to see the rest of the 255 colors.

Red, Green and Blue

Each file descriptor opened as an output device has a software color table associated with it. If multiple file descriptors are open to the same device, the software color table and the device's color map may not always be identical. The color table does not track the color map if the hardware color map is changed by another file descriptor path. For Starbase procedures that have parameters for red, green and blue, it is the color table that is searched for the closest color.

It is usually more efficient to select a color with an index rather than specifying a color with red, blue and green values due to the time it takes for the driver to figure out which pen in the color table most closely matches the specified color.

Selecting a color with the non-index version procedure will allow dithering for filled areas which is desirable in some cases.

Starbase Functionality

Commands Not Supported

The following commands are not supported and will be ignored:

```
bank_switch                  intline_width
backface_control             interior_style (INT_OUTLINE)
bf_control                   interior_style (INT_POINT)
bf_fill_color                light_ambient
bf_interior_style            light_attenuation
bf_perimeter_color           light_model
bf_perimeter_repeat_length   light_source
bf_perimeter_type            light_switch
bf_surface_coefficients      line_endpoint
bf_surface_model             pattern_define
define_trimming_curve        shade_range
depth_cue                    surface_model
depth_cue_color              surface_coefficients
depth_cue_range              viewpoint
hidden_surface               zbuffer_switch
```

Conditionally Supported

The following commands are supported under the listed conditions:

`interior_style`	Only the `INT_SOLID`, `INT_HATCH`, and `INT_HOLLOW` styles are supported.
`text_precision`	Only `STROKE_TEXT` precision is supported.
`shade_mode`	The color map mode may be selected but shading can not be turned on.
`vertex_format`	The use parameter must be zero; any extra coordinates supplied will be ignored.
`block_read`, `block_write`	The raw parameter for the `block_read` and `block_write` commands is normally ignored by this device driver. To use the `raw` mode, you must call the `gescape` function `R_BIT_MODE` which is discussed in the appendix of this manual.

Fast Alpha and Font Manager Functionality

This device driver supports raster text calls from the fast alpha and font manager libraries. See the *Fast Alpha/Font Manager's Programmer's Manual* for further information.

Parameters for gescape

The following `gescape` functions are common to many of the HP displays supported by Starbase. Detailed information about these functions can be found in Appendix A.

- `SWITCH_SEMAPHORE`—Semaphore control.
- `READ_COLOR_MAP`—Read color map.
- `BLINK_DISPLAY`—Blink display (blink rate is 2.4 Hz for this device).
- `R_GET_FRAME_BUFFER`—Read frame buffer address.
- `R_FULL_FRAME_BUFFER`—Full frame buffer.
- `R_LOCK_DEVICE`—Lock device.
- `R_UNLOCK_DEVICE`—Unlock device.
- `R_BIT_MODE`—Bit mode.
- `R_BIT_MASK`—Bit mask.
- `R_DEF_FILL_PAT`—Define fill pattern.

Performance Tips

The procedure `block_move` is faster when the block has a width and height that are a multiple of four, located on an origin that is a multiple of four from the upper left corner of the display. This performance increase is seen since the hardware block mover can be used on 4×4 boundaries, otherwise software routines must be used.

Cautions

The following cautions are provided in using this driver:

1. As mentioned previously, accessing the off-screen portion of the frame buffer (using `gescapes`) should be done with care, since other processes access this region. The guidelines for using this area are:
   - If you're using Starbase without Windows/9000, all off-screen frame buffer is available except the first and last five lines.
   - If you're using Starbase with Windows/9000, refer to the section "HP Windows Device Driver" for the window driver's utilization of the off-screen frame buffer. Again, Starbase still uses the bottom five lines of the frame buffer.
2. `SWITCH_SEMAPHORE` should be used with caution since it bypasses protection mechanisms used to prevent multiple processes from interfering with each other. For example, since the hardware resources can only be rationally used by one graphics process at a time, the driver activates a semaphore and locks the device before doing any output. This ensures, for example, that process A will not change the replacement rule while process B is in the middle of filling a polygon. It also prevents the terminal (`tty`) driver from overwriting any graphics processes that are outputting to the device. The driver unlocks the device when done processing output. Great caution should be used with this `gescape`.
3. Make sure when using `R_FRAME_BUFFER` that `R_LOCK_DEVICE` and `R_UNLOCK_DEVICE` bracket direct frame buffer accesses. Also make sure

there are no intervening Starbase attributes or primitive calls. Failure to follow this protocol can result in bus errors.

Contents

HP98710

The HP 98710 Device Driver

Device Description

Two device drivers are provided to access the HP 98700 Display:

- HP 98700—used to access the graphics display without using the optional graphics accelerator.
- HP 98710—used to access the graphics display using only the optional graphics accelerator.

This section covers the HP 98710 Device Driver; see the "HP 98700 Device Driver" section for information on using those devices.

The HP 98700H Graphics Display Station includes a high-resolution 19-inch color display, a display controller, an optional keyboard, and an optional graphics accelerator the HP 98710. The accelerator plugs into the bottom of the HP 98700 and provides additional performance. This driver provides an interface to the HP 98700H Graphics Display Station utilizing the optional graphics accelerator. For details on using the HP 98700H Graphics Display Station without the graphics accelerator, refer to the "HP 98700 Device Driver" section.

The interface for this device plugs into an I/O slot of supported SPUs. See table 1-8 in the introduction to this manual for the SPUs which support this device.

The display has a resolution of 1024×768 pixels. The color display system comes standard with four planes of frame buffer to provide 16 simultaneous colors. An optional four additional planes of frame buffer may be installed providing 256 simultaneous colors. A color map provides 8 bits per color (for red, green and blue), providing a color palette of over 16 million colors.

The display system is a bit-mapped device with special hardware and microcode for:

- write enabling planes
- displaying planes
- writing pixels to the frame buffer with a given replacement rule (see `drawing_mode` in *Starbase Reference* manual)
- blinking planes
- moving a block of pixels from one place in the frame buffer to another
- scan conversion (vector to raster) and clipping
- two-dimensional and three-dimensional transformations
- polygon clipping and filling

The display is organized as an array of bytes, with each byte representing a pixel on the display. With four planes installed, the 4 Least Significant Bits (LSBs) of each byte determine the color, providing color values from 0–15. When eight planes are installed, color values range from 0–255. These values are used to address the color map. The color map is basically a RAM table that has 16 or 256 addressable locations and is 24 bits wide (8 bits each for red, green and blue). Thus, the pixel value in the frame buffer addresses the color map, generating the color programmed at that location.

Typically, the user does not need to directly read or write pixels in the frame buffer. However, for those applications which require direct access, Starbase does provide the `gescape` function `R_GET_FRAME_BUFFER` which returns the virtual memory address of the beginning of the frame buffer (this `gescape` is discussed in the appendix). Frame buffer locations are then addressed relative to the returned address. The first byte of the frame buffer (byte 0) represents the upper left corner pixel of the screen. Byte 1 is immediately to its right. Byte 1023 is the last (right-most) pixel on the top line. Byte 1024 is the first (left-most) pixel on the second line from the top. The last (lower right corner) pixel on the screen is byte number 786,431 (767×1024+1023).

The actual frame buffer is 1024 bytes by 1024 bytes. The last 256 lines of the frame buffer are not displayed and are used for temporary storage of graphical items. This off-screen portion of the frame buffer may be accessed via the `gescape` function `R_FULL_FRAME_BUFFER` also documented in the appendix. Take care

when using this `gescape` since other processes and Starbase access the frame buffer off-screen memory. For example, Starbase uses the first line of off-screen memory and a 64×128 area with an upper left corner at (960,896).

Note The optional graphics accelerator is not used by HP Windows/9000 or the X Window systems. In addition, output from the HP 98710 driver may not be directed to a window in the HP Window/9000 or the X Window systems. However, the HP 98710 driver may open the `raw` display device while the Hewlett-Packard window systems are running. This allows the graphics accelerator to be used to write to the screen (while the Hewlett-Packard windows are running). An example demonstrating the HP 98710 driver used in this way is given near the end of this section.

Note The HP 98710 driver does not support multiple `gopens` of the same physical device.

Setting Up the Device

Switch Settings

On the Series 300 the HP 98700 and HP 98710 may be configured as an external display. This is done by setting switch two (SW-2) on the HP 98287A interface card. SW-2 is the eight-switch group. Place the interface card on an anti-static surface with the external connector and dust cover plate *away from you* and the bus connection pads *toward you.* This orientation is the same as the example shown below. If the third switch from the left is set to 1 (set toward you), the next five switches give the binary value of the HP 98700 and HP 98710's select code. If the third switch is set to 0, the HP 98700 and HP 98710 is an internal device (the default condition). For example, the following figure shows the switch settings necessary for the HP 98700 and HP 98710 to be used as an external device, at select code 25.

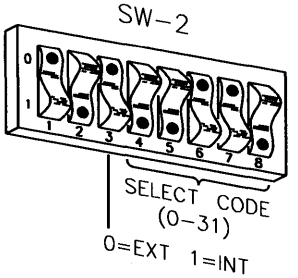


Figure HP98710-1. Switch Settings

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 `crt`.

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

When the device is at an internal address, see "Switch Settings", the `mknod` parameters are the character device with a major number of 12 and a minor number of 0 as in the following.

```
mknod /dev/crt c 12 0x000000
```

When the device is at an external address, see "Switch Settings", the `mknod` parameters are the character device with a major number of 12 and a minor number of `0x`⟨*sc*⟩`0200` where ⟨*sc*⟩ is the two-digit external select code. Note, the leading `0x` causes the number to be interpreted hexadecimally.

```
mknod /dev/crt  c  12  0x⟨sc⟩0200
```

Linking the Driver

The HP 98710 Device Driver is located in the `/usr/lib` directory with the file name `libdd98710.a`. This device driver may be linked to a program using the absolute path name `/usr/lib/libdd98710.a`, an appropriate relative path name, or by using the `-l` option `-ldd98710`. For example: to compile and link a program for use with this driver, use:

```
cc example.c -ldd98710 -lsb1 -lsb2 -o example
fc example.f -ldd98710 -lsb1 -lsb2 -o example
pc example.p -ldd98710 -lsb1 -lsb2 -o example
```

depending upon the language being used.

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/crt`.

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 is `hp98710` modified to meet the syntax of the programming language used, namely:

`"hp98710"`	*for C.*
`'hp98710'//char(0)`	*for FORTRAN77.*

`'hp98710'` *for Pascal.*

Mode — The mode control word consists 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.

- SPOOLED—cannot spool raster devices.
- 0—open the device, but do nothing else. The software color map is initialized on monochrome monitors.
- INIT—open and initialize the device as follows:
  1. Frame buffer is cleared to 0s.
  2. The color map is reset to its default values.
  3. The display is enabled for reading and writing.

Syntax Examples

To open and initialize an HP 98710 device for output:

For C Programs

```
fildes = gopen("/dev/crt",OUTDEV,"hp98710",INIT);
```

For FORTRAN77 Programs

```
fildes = gopen('/dev/crt'//char(0),OUTDEV,'hp98710'//char(0),INIT)
```

For Pascal Programs

```
fildes = gopen('/dev/crt',OUTDEV,'hp98710',INIT);
```

Special Device Characteristics

For device coordinate operations, location (0,0) is the upper-left corner of the screen with X-axis values increasing to the right and Y-axis values increasing down. The lower-right corner of the display is therefore (1023,767).

When the driver is opened, the microcode for the graphics accelerator is down loaded from the files located in `/usr/lib/starbase/hp98710`. Only one process may access (have opened with the **gopen** function) the HP 98710 device at a time.

Polygons of up to 255 vertices are supported. If a polygon has more than 255 vertices, only the first 255 vertices will be displayed.

Device Defaults

Number of Color Planes

When the `gopen` procedure is called, this driver asks the device for the number of color planes available. This number can be either 4 or 8. The device driver then acts accordingly.

Dither Default

The default number of colors searched for in a dither cell is 2. The number of colors allowed in a dither cell is 1, 2, 4, 8 or 16.

Raster Echo Default

The default raster echo is the 8×8 array:

255	255	255	255	0	0	0	0
255	255	0	0	0	0	0	0
255	0	255	0	0	0	0	0
255	0	0	255	0	0	0	0
0	0	0	0	255	0	0	0
0	0	0	0	0	255	0	0
0	0	0	0	0	0	255	0
0	0	0	0	0	0	0	255

The maximum size allowed for a raster echo is 64×64 pixels. The default drawing mode for the raster echo is 7 (a logical *OR*).

Color Planes Defaults

All planes are display-enabled. All planes are write-enabled.

Semaphore Default

Semaphore operations are enabled.

Line Type Defaults

The default line types are created with the bit patterns shown in the following table. The Starbase default line type is `SOLID`, line type `0`.

Table HP98710-1.

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

† Due to hardware differences, this line type is not the same for the HP 98700 Device Driver. Line type 4 is identical to line type 7 for the HP 98710 Device Driver.

Line Repeat Length

The *maximum* repeat length for lines is 1/3 the default VDC extent.

Default Color Map

If the fourth `gopen` parameter is zero (`0`), the current hardware color map is used on color displays.

If the fourth `gopen` parameter is `INIT`, the current color map is initialized to the default values in the following table.

Table HP98710-2. 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

Use the `inquire_color_table` procedure to see the rest of the 255 colors.

Red, Green and Blue

Each file descriptor opened as an output device has a software color table associated with it. If multiple file descriptors are open to the same device, the software color table and the hardware color map may not always be identical. The color table does not track the color map if the device's color map is changed by another file descriptor path. For Starbase procedures that have parameters for red, green and blue, it is the color table that is searched for the closest color.

It is usually more efficient to select a color with an index rather than specifying a color with red, blue and green values due to the time it takes for the driver to figure out which pen in the color table most closely matches the specified color.

Selecting a color with the non-index version procedure will allow dithering for filled areas which is desirable in some cases.

Starbase Functionality

Commands Not Supported (no-ops)

The following commands are not supported. They *will not* generate an error if they are called.

```
bank_switch                  interior_style (INT_OUTLINE)
backface_control             interior_style (INT_OUTLINE)
bf_control                   intline_width
bf_fill_color                light_ambient
bf_interior_style            light_attenuation
bf_perimeter_color           light_model
bf_perimeter_repeat_length   light_source
bf_perimeter_type            light_switch
bf_surface_coefficients      line_endpoint
bf_surface_model             pattern_define
define_trimming_curve        shade_range
depth_cue                    surface_model
depth_cue_color              surface_coefficients
depth_cue_range              viewpoint
hidden_surface               zbuffer_switch
```

Conditionally Supported

The following commands are supported under the listed conditions:

`interior_style`	Only the `INT_SOLID`, `INT_HATCH`, and `INT_HOLLOW` styles are supported.
`text_precision`	Only `STROKE_TEXT` precision is supported.
`shade_mode`	The color map mode may be selected, but shading can not be turned on.
`vertex_format`	The `use` parameter must be zero; any extra coordinates supplied will be ignored.

Vertices Per Polygon

The number of supported visible polygon vertices per polygon is 256. The hardware will truncate all vertices after the 256 limit.

Matrix Stack Limitations

The number of supported matrices in the matrix stack is 20. If the device driver is opened in `MODEL_XFORM` mode, the number of supported matrices in the matrix stack is only 18. The hardware will ignore matrices pushed on the stack after the stack is full.

Block_read, Block_write

The `raw` parameter for the `block_read` and `block_write` commands is normally ignored by this device driver. To use the `raw` mode, you must use the `gescape` function `R_BIT_MODE` discussed in the appendix.

Fast Alpha and Font Manager Functionality

This device driver does *not* support raster text calls from the fast alpha and font manager library.

Parameters for gescape

The following gescape functions are common to many of the Hewlett-Packard displays supported by Starbase. Detailed information about these functions can be found in the appendix.

- SWITCH_SEMAPHORE—semaphore control
- READ_COLOR_MAP—read color map
- BLINK_DISPLAY—blink display (blink rate is 2.4 Hz for this device.)
- R_GET_FRAME_BUFFER—read frame buffer address
- R_FULL_FRAME_BUFFER—full frame buffer
- R_LOCK_DEVICE—locks device
- R_UNLOCK_DEVICE—unlocks device
- R_BIT_MODE—bit mode
- R_BIT_MASK—bit mask
- R_DEF_FILL_PAT—define fill pattern

Performance Tips

Much higher performance can be obtained by using large polylines with this device rather than using move/draw procedures. Also, turning off semaphores yields a significant improvement in performance.

It should be noted that very little time is spent in Starbase when a polyline or polygon procedure is called for this device. Maximum performance will only be reached if the application program also takes very little time in preparing the data to send to Starbase.

The procedure `block_move` is faster when the block has a width and height with a multiple of four and are located on an origin with a multiple of four from the upper left corner of the display. This performance increase is seen since the hardware block mover can be used on 4×4 boundaries, otherwise software routines must be used.

Example Program - Using 98710 with Windows

```
****************
    wgade.c
****************
/*
This is a Starbase graphics program that accesses
HP Windows using the HP 98710 graphics accelerator.  the
key points are that the window must always be unobscured,
the program is actually accessing the raw device,
but is keeping it operations within a graphic window.
*/
#define TRUE 1
#define FALSE 0

#include <fcntl.h>
#include <stdio.h>
#include <errno.h>
#include <starbase.c.h>
extern int errno;
```

```
static int max_pen=255;
static int winx,winy;
float rawxm, rawym, res_x, res_y;

/*

Name: Lock device
Purpose: This procedure is used to lock the window, and make sure
       it is unobscured.  It is called before each Starbase call.
       The window needs to be unobscured because the 98710 driver
       does not understand about the obscured portion.  Once it is
       locked, this is the only process that can access the
       device.

Algorithm:
   Get the window's current position and sizes
   Lock the window
   While the window is obscured Do
      Unlock the window (so wm can change it)
      If the window is an icon, 'un'icon it
      If the window is not on top, move it on top
      (The next 3 operations can be done 2 different ways,
       either make the window completely visible, or set the
       clip rectangle to the current window boundaries, I
       chose the latter)
      If the viewport is not the upper left corner, move it there
      If the window is partially off screen, move it back on
      If the window is not full size, make it full size
      Get the current position, and sizes
      Lock the window
   Done While
   Turn off semaphores
   Set P1 & P2 to the new window position
```

```
Options: Instead of using wpan, wmove and wsize to make the
       window full size, you could use w, h, dx, dy and the
       dc_to_vdc Starbase call (using rawdev) to set the
       clipping rectangle.  This will cause the window to
       work much like a non-retained window (nothing will
       be written into obscured areas).  When using this
       option, do not use a R_GET_WINDOW_INFO since the
       window (when at a smaller size) will be obscured by
       its self.

Bugs: One problem with this procedure is that you cannot tell
     if a 'softkey' bar is causing the window to be obscured.
     It would be possible to determine the limit by trial and
     error (not in a program), at using that as the maximum y
     setting (ym).
*/
LOCK_device(rawdev,sbwdev)
int rawdev,sbwdev;
{

   int x,y,w,h,dx,dy,rw,rh;
   int arg1[2],arg2[2];
   float p1_x,p1_y,p1_z,p2_x,p2_y,p2_z;

   wgetcoords(sbwdev,&x,&y,&w,&h,&dx,&dy,&rw,&rh);
   arg1[0]=1; /* must be 1 to pick up the sprite if necessary */
   gescape(sbwdev,R_LOCK_DEVICE,arg1,arg2);
   gescape(sbwdev,R_GET_WINDOW_INFO,arg1,arg2);
```

```
    while (arg2[1]==0) {
       arg2[1]==1;
       gescape(sbwdev,R_UNLOCK_DEVICE,arg1,arg2);

       if (wiconic(sbwdev,-1))
          wiconic(sbwdev,0);

       if (!wtop(sbwdev,-1))
          wtop(sbwdev,1);

       if (dx!=0 | dy!=0)
          wpan(sbwdev,0,0);

       if (x+rw > rawxm)
          if (y+rh>rawym)
             wmove(sbwdev,(int)rawxm-rw,(int)rawym-rh);
          else
             wmove(sbwdev,(int)rawxm-rw,y);

       if (w<rw | h<rh)
          wsize(sbwdev,rw,rh);
       wgetcoords(sbwdev,&x,&y,&w,&h,&dx,&dy,&rw,&rh);
       arg1[0]=1;
       gescape(sbwdev,R_LOCK_DEVICE,arg1,arg2);
       gescape(sbwdev,R_GET_WINDOW_INFO,arg1,arg2);
    }
    arg1[0]=0;
    gescape(rawdev,SWITCH_SEMAPHORE,arg1,arg2);
    gescape(sbwdev,SWITCH_SEMAPHORE,arg1,arg2);

    if (x!=winx | y!=winy) {
       winx=x;
       winy=y;
       p1_x=x*res_x;
       p1_y=(rawym-(y+rh))*res_y;
       p1_z=0.0;
       p2_x=(x+rw)*res_x;
       p2_y=(rawym-y)*res_y;
       p2_z=1.0;
       set_p1_p2(rawdev,METRIC,p1_x,p1_y,p1_z,p2_x,p2_y,p2_z);
    }
}
```

```
/*
Name: Unlock device
Purpose: Unlock the window display.

Algorithm:
   Flush the Starbase buffer
   Turn on semaphores
   Unlock the device
*/
UNLOCK_device(rawdev,sbwdev)
int rawdev,sbwdev;
{

   int x,y,w,h,dx,dy,rw,rh;
   int arg1[2],arg2[2];

   make_picture_current(rawdev);
   arg1[0]=1;
   gescape(rawdev,SWITCH_SEMAPHORE,arg1,arg2);
   gescape(sbwdev,SWITCH_SEMAPHORE,arg1,arg2);
   gescape(sbwdev,R_UNLOCK_DEVICE,arg1,arg2);
}
/*
Name: Main
Purpose: Main program to do graphics to the screen.

Algorithm:
   Open the window manager
   Create a window
   Open the graphics window
   Open the raw device
   Move the window to the top
   Set clear control to vdc limits (can't clear to
      device limits because we will clear the full
      window system)
   Get the raw device size
   Get the window device size (for graphics)
   Lock the device (must be done before Starbase calls)
   Set various options
   Clear the device
   Unlock the device (so other processes can do things)
   Do some polygons (using LOCK_device and UNLOCK_device)
   Do some vectors (using LOCK_device and UNLOCK_device)
   Do some text and rectangles (using LOCK_device and UNLOCK_device)
   close the window and raw devices
```

```
Required: The only required operations are to open both the window and
         the raw device.  Device limits should be based on the window,
        with the raw limits used by set_p1_p2 in the LOCK_device
        procedure.  Clear control should be either CLEAR_VDC_EXTENT
        or CLEAR_CLIP_RECTANGLE never CLEAR_DISPLAY_SURFACE. You
        must do a winit on the window, so 'w' calls work. You should
        do LOCK_device and UNLOCK_device around each Starbase call
        that accesses the device.  Some calls don't access the device,
        such as character_height, or clear_control, but when in doubt
        it is better to lock the device.  When using device coordinates
        remember to offset the values by the current window location.
*/

main(argc,argv)
int argc; char *argv[];
{
        int fildes,fildes2,wmfd;
        int curr_color,error,mode=0;
        int n,loops=2;
      float xm,ym;
        float x,y,inc=40.0;
        float ri,imax;
        float pts[20];
        float *ptr;
        int has_edges;
        float p1[2][3],res[3],p1[3],p2[3];
       char wmname[100];
      wmpathmake("WMDIR", "wm", wmname);
      wmfd = open(wmname, O_RDWR);
      if (wmfd == -1)
         perror("Can't open wm"), exit(1);
      if (winit(wmfd))
         perror("Can't winit wm"), exit(1);
```

```
winx=(-1);winy=(-1);
error = wcreate_graphics(wmfd,"/dev/screen/gwindow",150,100,\
                         500,400,500,400,1,1);
fildes2 = gopen("/dev/screen/gwindow",OUTDEV,"hp98700",mode);
fildes = gopen("/dev/gcrt",OUTDEV,"hp98710",mode);

/*fildes2 = gopen("/dev/screen/gwindow",OUTDEV,"hp300h",mode);
fildes = gopen("/dev/crt",OUTDEV,"hp300h",mode);*/
if (fildes2 == -1)
   fprintf(stderr,"could not open window device\n");
if (fildes == -1)
   fprintf(stderr,"could not open raw device\n");
if (winit(fildes2))
   perror("Can't winit fildes2"), exit(1);
if (wtop(fildes2, 1) == -1)
   perror("Could not wtop fildes2"), exit(1);

clear_control(fildes,CLEAR_VDC_EXTENT);
mapping_mode(fildes,DISTORT);

  inquire_sizes(fildes,pl,res,p1,p2,&max_pen);
if (pl[1][1]==0.0)  pl[1][1]=pl[0][1];
  rawxm=pl[1][0];       rawym=pl[1][1];
  inquire_sizes(fildes2,pl,res,p1,p2,&max_pen);
if (pl[1][1]==0.0)  pl[1][1]=pl[0][1];
  xm=pl[1][0];        ym=pl[1][1];
res_x=res[0];
res_y=res[1];
LOCK_device(fildes,fildes2);
  vdc_extent(fildes,0.0,0.0,0.0,xm,ym,0.0);
  clip_rectangle(fildes,0.0,xm,0.0,ym);
  imax = (xm > ym ? ym / 2 : xm / 2);

  background_color_index(fildes,2 % max_pen);
  clear_view_surface(fildes);
/* TRY POLYGONS */
interior_style(fildes,INT_SOLID,TRUE);
UNLOCK_device(fildes,fildes2);
  for (n=0;   n < loops;   n++) {
   LOCK_device(fildes,fildes2);
   make_picture_current(fildes);
   clear_view_surface(fildes);
   curr_color = 0;
```

```
perimeter_color_index(fildes,1);
UNLOCK_device(fildes,fildes2);
has_edges = FALSE;
for (x=5;   x<=(xm+inc);   x=(x+2*inc)) {
      if (x>xm/2) has_edges = TRUE;
      for (y=5;   y<=(ym+inc);   y=(y+2*inc)) {
        ptr = &pts[0];
        *ptr++ = x;       *ptr++ = y+inc;
        if (has_edges) *ptr++ = TRUE;
        *ptr++ = x+inc;   *ptr++ = y;
        if (has_edges) *ptr++ = FALSE;
        *ptr++ = x;       *ptr++ = y-inc;
        if (has_edges) *ptr++ = TRUE;
        *ptr++ = x-inc;   *ptr++ = y;
        if (has_edges) *ptr++ = FALSE;
        *ptr++ = x;       *ptr++ = y+inc;
        if (has_edges) *ptr++ = TRUE;
          LOCK_device(fildes,fildes2);
        fill_color_index(fildes,(curr_color++) % max_pen);
        polygon2d(fildes,pts,4,has_edges);
          UNLOCK_device(fildes,fildes2);
      }
}
sleep(5);
curr_color = 0;
has_edges = FALSE;
if (n) {
   LOCK_device(fildes,fildes2);
   clear_view_surface(fildes);
   UNLOCK_device(fildes,fildes2);
}
for (x=5;   x<=(xm+inc);   x=(x+2*inc)) {
      if (x>xm/2) has_edges = TRUE;
      for (y=5;   y<=(ym+inc);   y=(y+2*inc)) {
        ptr = &pts[0];
        *ptr++ = x;       *ptr++ = y+inc;
        if (has_edges) *ptr++ = FALSE;
        *ptr++ = x+inc;   *ptr++ = y;
        if (has_edges) *ptr++ = TRUE;
        *ptr++ = x;       *ptr++ = y-inc;
        if (has_edges) *ptr++ = FALSE;
        *ptr++ = x-inc;   *ptr++ = y;
```

```
                if (has_edges) *ptr++ = TRUE;
                *ptr++ = x;        *ptr++ = y+inc;
                if (has_edges) *ptr++ = FALSE;
                LOCK_device(fildes,fildes2);
                if (!has_edges)
                    line_color_index(fildes,(curr_color++) %
max_pen);
                else
                    background_color_index(fildes,2 % max_pen);
                polyline2d(fildes,pts,5,has_edges);
                UNLOCK_device(fildes,fildes2);
              }
          }
          sleep(5);
        }   /*for n*/
      background_color_index(fildes,6 % max_pen);
      for (n = 0; n < loops; n++) {
            LOCK_device(fildes,fildes2);
            clear_view_surface(fildes);
            UNLOCK_device(fildes,fildes2);
            for (ri = 0; ri <= imax; ri++) {
                  LOCK_device(fildes,fildes2);
                  line_color_index(fildes,(int)ri % max_pen);
                  move2d(fildes,ri, ri);
                  draw2d(fildes,xm-ri, ri);
                  draw2d(fildes,xm-ri, ym-ri);
                  draw2d(fildes,ri, ym-ri);
                  draw2d(fildes,ri, ri);
                  UNLOCK_device(fildes,fildes2);
            }
      }   /*for n*/
      LOCK_device(fildes,fildes2);
      clear_view_surface(fildes);
      character_height(fildes,20.0);
      character_width(fildes,15.0);
      UNLOCK_device(fildes,fildes2);
      for (y=0;   y<ym-inc;   y=y+inc) {
            LOCK_device(fildes,fildes2);
            rectangle(fildes,y-5.0,y-5.0, y+(4.0*inc),y+(2.0*inc));
            text_color_index(fildes,(int)(y+1) % max_pen);
            text2d(fildes,y,y,"abcdefghijkxyzABC",0,0);
            UNLOCK_device(fildes,fildes2);
      }
      sleep(6);
        gclose(fildes);
```

```
        gclose(fildes2);
          exit(0);
}
```

Cautions

The following cautions are provided in using this driver:

1. As mentioned previously, accessing the off-screen portion of the frame buffer (using the `gescape` function) should be done with care, since other processes access this region.

2. If the HP 98710 is configured in external I/O space (SW-2 switch at pin 3 set to 0 on HP 98287 interface) the graphics accelerator should be initialized (after power-up) before a program using the HP 98700 driver or HP Windows/9000 with the HP 98700 driver is run to that device. The following method is recommended for HP 98710 users:

   Enter following program:

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

main(argc,argv)
int argc;
char *argv[];
{
  int fildes;
  if (argc > 1) {
    fildes = gopen(argv[1],OUTDEV,"hp98710",INIT);
    if (fildes == -1) exit(-1);
    gclose(fildes);
 }
 else printf("ERROR:user must specify device file\n");
}
```

3. Store program as `te_init.c` and execute

```
cc -o te_init te_init.c -ldd98710 -lsb1 -lsb2
```

4. Move `te_init` to `/usr/lib/starbase/hp98710` (Consult system administrator to do this.)

5. Edit `/etc/rc` file and insert following line:

```
/usr/lib/starbase/hp98710/te_init /dev/crt
```

6. Be sure to use the correct device file name for the HP 98710. The above example uses `crt` as an example device file name. Consult your system administrator for the name that is applicable to your system.

7. Make sure the `R_LOCK_DEVICE` and `R_UNLOCK_DEVICE` **gescape** functions bracket direct frame buffer accesses when the `R_GET_FRAME_BUFFER` **gescape** is used. Make sure there are no intervening Starbase attribute or primitive calls. Failure to follow this protocol can result in bus errors.

Contents

HP98720

The HP 98720 Device Driver

Device Description

The graphics display station includes a high resolution 19 inch color display, an HP 98720A Display Controller, and an optional graphics accelerator. The display controller plugs into an I/O slot of the SPUs. (See the "Introduction" section of this manual for systems supporting this controller.)

Three device drivers are provided to access the HP 98720 display:

- HP 98720—used to access graphics windows with the X Window system or the graphics display without using the optional graphics accelerator.
- HP 98720w—used to access graphics windows with HP Windows/9000, or the graphics display without windows. The latter is useful for doing fast alpha or font manager operations to the graphics display.
- HP 98721—used to access the graphics display using only the optional graphics accelerator.

This section covers the HP 98720 Device Driver, see the "HP 98720w Device Driver" or the "HP 98721 Device Driver" sections in this manual for information on using those device drivers.

The display has a resolution of 1280 by 1024 pixels. The standard color display system has four planes of frame buffer to provide 16 simultaneous colors. You can add optional memory in banks of eight planes each. A fully configured system consists of three banks of frame buffer for full 24 bit per pixel color, one bank for full Z-buffer capability (with graphics accelerator), and four overlay planes for non-destructive alpha, cursors, or graphics.

An 8-plane configuration allows 256 colors to be displayed simultaneously from a palette of 16 million. A 16 plane system is like two 8-plane frame buffers where only one 8-plane buffer is displayed. This configuration is useful for double buffering. When three banks of frame buffer are installed, the system may be

configured to display eight bits red, eight bits green, and eight bits blue per pixel. Double buffering may also be achieved at a resolution of four bits red, four bits green, and four bits blue.

The display system is a bit-mapped device with special hardware for:

- Write enable/disable individual planes.
- Video enable/disable individual planes.
- Memory writes with specified replacement rule. (see `drawing_mode`)
- Video blinking of individual planes.
- Video blinking of individual color map locations.
- Arbitrary sized rectangular memory to memory copies.

The display is organized as an array of bytes, with each byte representing a pixel on the display. With four planes installed, the four least significant bits of each byte determine the color, providing color map indices from 0-15. When eight planes are installed, color map indices range from 0-255. The color map is a RAM table that has 16 or 256 addressable locations and is 24 bits wide (8 bits each for red, green, and blue). Thus, the pixel value in the frame buffer addresses the color map, generating the color programmed at that location.

If you add optional banks of frame buffer memory to the minimal system, the four standard image planes function as overlay planes. These overlay planes have their own unique color map, separate from the color map used for the newly installed image planes. This color map consists of sixteen 4-bit entries. These four bits correspond to transparent, red, green, and blue (in order of Most Significant Bit (MSB) to Least Significant Bit (LSB). If the transparent bit (the MSB) is set to zero, the pixel color will be the color of the image 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 primary colors or transparent.

You can use overlay planes for non-destructive alpha, graphics, or cursors. For example, when the HP 98720 is used on the system console, the Internal Terminal Emulator (ITE) uses three of the overlay planes for alpha information. This way there is no interaction between ITE text and images in the graphics planes. Windows/9000 also runs in the overlay planes. Refer to the "HP 98720w Device Driver" section of this manual for more information. The X Window system

runs in configurations involving both image and overlay planes. See the *Starbase Programming with X11* manual for more information. To do graphics in the overlay planes the HP 98720 Device Driver may be opened directly to the overlay planes as if they were a separate device. Refer to the segment "Setting up the Device" in this section for more information.

One overlay plane is reserved for graphic cursors. When Starbase cursors are in the overlay plane performance is enhanced, since it it not necessary to "pick up" the cursor each time the frame buffer is updated. You can think of the overlay plane used for cursors as a separate cursor plane. Any data in the cursor plane will be displayed over data in the graphics planes. Data in the other three overlay planes will be displayed over data in the graphics planes and the cursor plane. For example, suppose a graphics application is running in the graphics planes while the window manager is running. If the application has a Starbase cursor in the overlay cursor plane, the cursor will always be visible inside regions of see-thru because the cursor has display priority over the graphics. (Refer to the "HP Windows/9000 See-Thru Color" section in the "HP 98720w Device Driver".) If the cursor is moved outside the graphics window boundary, it is not visible since the window desktop environment is drawn to the overlay planes, which have display priority over the cursor plane.

Typically, the user does not need to directly read or write pixels in the frame buffer. However, for those applications which require direct access, Starbase does provide the `gescape` function `R_GET_FRAME_BUFFER`, which returns the virtual memory address of the beginning of the frame buffer. This `gescape` is discussed in the appendix of this manual. Frame buffer locations are then addressed relative to the returned address. The first byte of the frame buffer (byte 0) represents the upper left corner pixel of the screen. Byte 1 is immediately to its right. Byte 1279 is the last (right-most) pixel on the top line. The next 768 bytes of the frame buffer are not displayable. Byte 2048 is the first (left-most) pixel on the second line from the top. The last (lower right corner) pixel on the screen is byte number 2,096,383.

If more than one bank of optional frame buffer is installed, bank switching must be used to access the additional memory. A number of Starbase calls may set the bank register so it is advisable to call `bank_switch` just prior to making accesses to the frame buffer pointer to ensure desired results.

The off-screen portion of the frame buffer may be accessed via the **gescape** function R_FULL_FRAME_BUFFER. Use this **gescape** carefully since other processes, Starbase, and Windows/9000 access the frame buffer off-screen memory.

Series 800

On the Series 800 computers, a write to I/O space must be on the word boundaries. The frame buffer is mapped as integer (32bits) per pixel. Therefore, when you write directly to the frame buffer on the HP 98720 Graphics Display Station, each pixel is written with an integer access.

If writing to the HP 98720 image buffer and not in CMAP_FULL color map mode, only one bank can be written at a time. The bank to be written must be established by a call to **bank_switch**. Then, to write to bank *n*, place the pixel value to be written into byte *n* of the interger, where *n* can be 0, 1, or 2, and the LSB is byte 0.

All three banks for one pixel can be written simultaneously by packing all three bank values for the pixel into the integer value and having the color map mode as CMAP_FULL before writing.

Setting Up the Device On Series 300

The HP 98720 Device Driver can be used if the display is configured in either internal or external address space. Refer to the *Configuration Reference Manual* for a description of internal and external address space.

Note If the HP 98720 is configured as an external display, there will not be an Internal Terminal Emulator (ITE) for that device. Since it is the ITE that normally initializes the display, external devices must be reset after power-up by running a simple Starbase program with a mode of `RESET_DEVICE` in the `gopen` call. It may also be necessary to run this program after running an application which manipulated the overlay color map, such as windows. An example program, which could be called from `/etc/rc` during power-up, is given at the end of this section. For more details concerning the effects of `RESET_DEVICE`, see the "Device Initialization" segment of this section.

Switch Settings

The Graphics Interface card has a single 6-bit address select switch. One bit, labeled `FB`, determines the frame buffer location, while the other five switch bits, labeled `CS`, determine the location within the DIO memory map of the HP 98720 control space. Silkscreening on the printed circuit board indicates the meaning of the bits.

The frame buffer consumes two Mbytes of I/O address space, starting at `FB_BASE`. The switch bit labeled `FB` determines the address of `FB_BASE` as shown below.

Table HP98720-1. HP 98720 Frame Buffer Locations

FB	FB_BASE (hex)
0	$200000
1	$800000

Typical systems will map the frame buffer to $200000. However, some systems which have multiple displays may map the frame buffer address to $800000.

When the frame buffer address is set to $800000, the HP Series 300 Model 320 SPU memory limit is reduced from 7.50 Mbytes to 5.75 Mbytes. This occurs since the frame buffer is mapped into the upper two Mbytes of memory address space.

The control space requires 128 Kbytes starting at CTL_BASE. The five switch bits labelled CS, determine the address of CTL_BASE. The HP 98720 may be configured as an external or internal display. Since only 64 Kbytes are normally allotted for external I/O select codes, two consecutive select codes will be consumed if the device is configured as an external display. The control space may be located at any of 32 positions. Sixteen positions are reserved in internal I/O space and sixteen are in external I/O space (with five reserved). The table below lists the binary switch setting with the corresponding values of CTL_BASE for external I/O settings, as well as the select code spaces consumed.

Table HP98720-2. Control Space Settings (External I/O)

CS Setting	CTL_BASE (hex)	Select Codes
01011	$560000	
10101	$6A0000	10-11
10110	$6C0000	12-13
10111	$6E0000	14-15
11000	$700000	16-17
11001	$720000	18-19
11010	$740000	20-21
11011	$760000	22-23
11100	$780000	24-25
11101	$7A0000	26-27
11110	$7C0000	28-29
11111	$7E0000	30-31

If the HP 98720 is configured as the system console, CTL_BASE needs to be placed at $560000, which is an internal I/O setting. If the device is not used as the system console, the control space should not be placed in internal I/O space, since it is likely to overlap the address space of other system hardware. In this case, an external I/O space setting should be selected with two consecutive select codes which are unused by the system.

Example Program To Reset HP 98720

```
/*
 * Starbase program: reset98720.c
 * Compile: cc -o reset98720 reset98720.c -ldd98720 -lsb1 -lsb2
 * Destination: /usr/bin
 * Execute: add line to the /etc/rc - "/usr/bin/reset98720 /dev/crt.external"
 *
 * Example program to be put in /etc/rc for resetting an external HP 98720
 * device during power-up.
 */
#include <starbase.c.h>

main(argc,argv)
int argc; char *argv[];
{
   int fildes;

   if ((fildes = gopen(argv[1],OUTDEV,"hp98720",INIT|RESET_DEVICE)) < 0)
        printf("External HP 98720 %s initialization failed.\n",argv[1]);
   else
      printf("External HP 98720 %s initialization succeeded.\n",argv[1]);
      gclose(fildes);
   }
}
```

Setting Up the Device On Series 800

Up to four HP 98720 devices can be connected to a Series 800 Model 825 or 835 SPU. However, it is recommended that only two HP 98720 devices have the Internal Terminal Emulator (ITE) or window systems running on them.

Only one HP 98720 device can be connected to a Series 800 Model 840.

Special Device Files (mknod) On Series 300

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 details. 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 the devices is `crt`.

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

When the device is at the internal address (refer to the "Switch Settings" section) the `mknod` parameters should create a character special device with a major number of 12 and a minor number of 0.

```
mknod /dev/crt c 12 0x000000
```

When the device is at an external address (refer to the "Switch Settings" section) the `mknod` parameters should create a character special device with a major number of 12 and a minor number of `0x`⟨*sc*⟩`200` where *Sc* is the two-digit external select code. Note that the leading `0x` causes the number to be interpreted hexidecimally.

```
mknod /dev/crt c 12 0x⟨sc⟩0200
```

The HP 98720 Device Driver may also be opened to the overlay planes in graphics mode if they are present. Since one plane is still reserved for cursors, the graphics device will look like a three plane device in this mode. Since the terminal emulator and window system operate in the overlay planes also, there will be interactions with these processes if a graphics driver is opened in this manner while these processes are present. To open the HP 98720 Device Driver to the overlay planes instead of the graphics planes, the last byte of the minor number must be 1.

For example, when the device is at an internal address, the `mknod` parameters for the overlay device should create a character special device with a major number of 12 and a minor number of 1.

```
mknod /dev/ocrt c 12 0x00001
```

When the device is at an external address (refer to the section on "Switch Settings") the `mknod` parameters for the overlay device should create a character special device with a major number of 12 and a minor number of 0x⟨*sc*⟩0201 where ⟨*sc*⟩ is the two-digit external select code.

```
mknod /dev/ocrt c 12 0x⟨sc⟩0201
```

Special Device Files (mknod) On Series 800

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 the *HP-UX Reference* for further details. 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 names that are suggested for the devices are `crt`, `crt0`, `crt1`, or `crt2`.

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

When creating the device file the `mknod` parameters should create a character special device with a major number of 14 and a minor number of the format below (where ⟨*lu*⟩ is the two digit hardware logical unit number). Note that the leading `0x` causes the number to be interpreted hexadecimally.

```
mknod /dev/crtx c 14 0x00⟨lu⟩00
```

The HP 98720 Device Driver may also be opened to the overlay planes in graphics mode if they are present. Since one plane is still reserved for cursors, the graphics

device will look like a three plane device in this mode. Since the terminal emulator and window system operate in the overlay planes also, there will be interactions with these processes if a graphics driver is opened in this manner while these processes are present. To open the HP 98720 Device Driver to the overlay planes instead of the graphics planes, the last byte of the minor number must be 1.

For example, the `mknod` parameters for the overlay device should create a character special device with a major number of 14 and a minor number of the format indicated below (where ⟨*lu*⟩ is the two digit hardware logical unit number):

```
mknod /dev/ocrtx c 14 0x00⟨lu⟩01
```

Linking the Driver

The HP 98720 Device Driver is the file `libdd98720.a in the /usr/lib` directory. This device driver may be linked to a program using the absolute path name `/usr/lib/libdd98720.a`, an appropriate relative path name, or by using the `-l` option `-ldd98720`. For example, to compile and link a program for use with this driver, use:

```
cc example.c -ldd98720 -lsb1 -lsb2 -o example
fc example.f -ldd98720 -lsb1 -lsb2 -o example
pc example.p -ldd98720 -lsb1 -lsb2 -o example
```

depending upon the language being used.

Device 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 (for example, `/dev/crt`.)

Kind — Indicates the I/O characteristics of the device. This parameter must be `OUTDEV` for this driver unless a window is being opened, in which case it may be `OUTINDEV`.

Driver — This is the character representation of the driver type. This must be hp98720. For example:

`"hp98720"`	*for C.*
`'hp98720'//char(0)`	*for FORTRAN77.*
`'hp98720'`	*for Pascal.*

Mode — The mode control word consisting of several flag bits which are *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. See the *Starbase Programming with X11* manual for a description of the **gopen** function of an X window.

- `SPOOLED`—cannot spool raster devices.
- `MODEL_XFORM`—Shading is not supported for this device. However, opening in `MODEL_XFORM` mode will affect how matrix stack and transformation routines are performed.
- `0`—open the device, but do nothing else. The software color map is initialized on monochrome monitors.
- `INIT`—open and initialize the device as follows:
  1. Clear frame buffer to 0s.
  2. Reset the color map to its default values.
  3. Enable the display for reading and writing.

- RESET_DEVICE—open and reset the device as follows:
    1. Clear frame buffer and overlays to 0s.
    2. Reset the color map to its default values.
    3. Clear the overlay color map.
    4. Enable the display for reading and writing.
    5. Reset the graphics accelerator.

  Note that the RESET_DEVICE flag bit should be used with caution: it will adversely affect any other processes using the device. This flag bit is intended to reset a device completely. This should only be necessary for devices in an unknown state such as a device powered up in an external I/O space. Most programs should not use this flag bit.

Syntax Examples

To open and initialize an HP 98720 device for output:

For C Programs:

```
fildes = gopen("/dev/crt",OUTDEV,"hp98720",INIT);
```

For FORTRAN77 Programs:

```
fildes = gopen('/dev/crt'//char(0), OUTDEV,'hp98720'//char(0),INIT)
```

For Pascal Programs:

```
fildes = gopen('/dev/crt',OUTDEV,'hp98720',INIT);
```

Special Device Characteristics

For Device Coordinate operations, location $(0,0)$ is the upper-left corner of the screen with X-axis values increasing to the right and Y-axis values increasing down. The lower-right corner of the display is therefore $(1279, 1023)$.

Offscreen Memory Usage

Each time the HP 98720 Device Driver is opened it allocates a portion of offscreen memory. This is used for fill pattern storage, raster echo definitions, and other functions. The size of the areas used are 64×192 pixels and 32×4 pixels. If the

driver has been opened to the overlay planes, the offscreen area used is in the overlay planes; otherwise the area used is in the graphics planes. Up to ten of these offscreen areas may be allocated. One is reserved for the HP 98721 Device Driver and the other nine are for HP 98720 Device Drivers. This means that no more than one HP 98721 Device Driver and nine HP 98720 Device Drivers may be opened to a device at the same time. If nine HP 98720 Device Drivers are opened to the graphics planes, another nine may be opened to the overlay planes. However, only one HP 98721 Device Driver may be opened to a device at any time to either graphics or overlay planes. The X11 server uses a similar cursor area and also uses offscreen for client pixmaps. Accessing offscreen memory while the X11 server is running is not recommended.

See the "HP 98720w Device Driver" and "HP 98721 Device Driver" sections for more information on offscreen memory usage.

Device Defaults

Number of Color Planes

When the `gopen` procedure is called, this driver asks the device for the number of color planes available. This number can be either 3 (for the overlay planes) or 4, 8, 16, or 24 (for the image planes). The device driver then acts accordingly.

Dither Default

The default number of colors searched for in a dither cell is 2. The number of colors allowed in a dither cell is 1, 2, 4, 8 or 16. For devices having 24 or more planes in `CMAP_FULL` mode (see `shade_mode`) dithering is not supported since full 24-bit color is available. If you are double buffering with 12 planes per buffer then the number of colors allowed in a dither cell is 1, 2, or 4.

Raster Echo Default

The default raster echo is the 8×8 array:

```
255 255 255 255   0   0   0   0
255 255   0   0   0   0   0   0
255   0 255   0   0   0   0   0
255   0   0 255   0   0   0   0
  0   0   0   0 255   0   0   0
  0   0   0   0   0 255   0   0
  0   0   0   0   0   0 255   0
  0   0   0   0   0   0   0 255
```

The maximum size allowed for a raster echo is 64×64 pixels. The default drawing mode for the raster echo is 7 (a logical *OR*).

By default the raster echo is written to the graphics planes. All other echo types are written to an overlay plane. The location of raster and non-raster echoes can be changed using the `gescape` function `R_OVERLAY_ECHO`.

Color Planes Defaults

The default configuration is a 4- or 8-plane color mapped system regardless of the number of frame buffer banks installed.

All planes in the first bank are display enabled. All planes in the first bank are write enabled.

Semaphore Default

Semaphore operations are enabled.

Line Type Defaults

The default line types are created with the bit patterns shown below:

Table HP98720-3.

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

Default Color Map

If the fourth **gopen** parameter is zero (0), the current hardware color map is used on color displays.

If the fourth **gopen** parameter is `INIT`, the current color map is initialized to the default values shown below.

Table HP98720-4. 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

Use the `inquire_color_table` procedure to see the rest of the 255 colors.

When `INIT` is used in the `shade_mode` procedure call the color map initialization is based on the value of the `mode` parameter and the number of frame buffer banks installed.

`mode=CMAP_NORMAL` Only one bank of the first two can be displayed at a time. If a third bank is installed it can not be displayed in this mode.

`mode=CMAP_MONOTONIC` The color map will be initialized as:

```
for (i=0; i<256; i++) {
  cmap[i].red = cmap[i].green = cmap[i].blue = i/255.0;
}
```

Only one bank of the first two can be displayed at a time. If a third bank is installed it can not be displayed in this mode.

`mode=CMAP_FULL`

With less than three banks installed the color map will be initialized as three bits red, three bits green and two bits blue. The three most significant bits are red and the two least significant bits are blue. Only one bank of the first two can be displayed at a time.

With three or more banks installed the color map will be initialized as the `CMAP_MONOTONIC` case above but now the first bank of eight will go through the blue portion of the color map, the second bank goes through the green portion, and the third bank goes through the red portion. In this mode the color map is transparent and the eight bits from each bank drives the appropriate video color level. The color map could be subsequently modified in this mode to perform functions like gamma correction or double buffering of four bits per color.

Red, Green, and Blue

Each file descriptor opened as an output device has a color table associated with it. If multiple file descriptors are open to the same device, the color table and the device's color map may not always be identical. The color table does not track the color map if the device's color map is changed by another file descriptor path.

For Starbase procedures that have parameters for red, green, and blue, the way the actual color is chosen depends on the current `shade_mode` setting.

`mode=CMAP_NORMAL`

The color map is searched for the color that is closest in RGB space to the one requested. That color map index is written to the frame buffer for subsequent output primitives. It is more efficient to select a color with an index rather than specifying a color with red, blue, and green values in this mode because it takes extra time to figure out which index in the color table most closely matches the specified color.

`mode=CMAP_MONOTONIC`

The red, green, and blue value is converted to an intensity value using the equation:

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

This intensity is converted to an index value by mapping intensity 0.0 to the minimum index set by `shade_range`, and intensity 1.0 to the maximum index set by `shade_range`. This mode is useful for displaying a high-quality monochrome picture on an 8-plane system from data that produces a high quality color picture on a 24-plane system.

`mode=CMAP_FULL` — The color values will be mapped directly to an index with the assumption the color map is setup to a predefined full color state.

Starbase Functionality

Commands Not Supported

The following commands are ignored.

```
backface_control              hidden_surface
bf_control                    interior_style (INT_OUTLINE)
bf_fill_color                 interior_style (INT_POINT)
bf_interior_style             light_ambient
bf_perimeter_color            light_attenuation
bf_perimeter_repeat_length    light_model
bf_perimeter_type             light_source
bf_coefficients               light_switch
bf_surface_model              shade_range
define_trimming_curve         surface_coefficients
depth_cue                     surface_model
depth_cue_color               viewpoint
depth_cue_range               zbuffer_switch
```

Commands Conditionally Supported

The following commands are supported under the listed conditions:

`pattern_define`	4×4 is the largest supported pattern.
`shade_mode`	The color map mode may be selected but shading can not be turned on.
`text_precision`	Only `STROKE_TEXT` precision is supported.
`vertex_format`	The `use` parameter must be zero, any extra coordinates supplied will be ignored.

block_read, block_write

The "`raw`" parameter for the `block_read` and `block_write` commands is normally ignored by this device driver. To use the raw mode, you must call the `gescape` function `R_BIT_MODE` discussed in the appendix of this manual.

Fast Alpha and Font Manager Functionality

The HP 98720 Device Driver supports raster text calls from the fast alpha and font manager libraries. These calls may be made while running in the overlay or image planes. See the *Fast Alpha/Font Manager's Programmer's Manual* for further information.

Parameters for gescape

The following gescape functions are common to two or more of the Hewlett-Packard displays supported by Starbase. Detailed information about these functions can be found in Appendix A.

- BLINK_INDEX—Alternate between HP 98720 hardware color maps.
- BLINK_PLANES—Blink display (blink rate is 3.75 Hz for this device).
- R_BIT_MASK—Bit mask.
- R_BIT_MODE—Bit mode.
- R_DEF_ECHO_TRANS—Define raster echo transparency.
- R_DEF_FILL_PAT—Define fill pattern.
- R_FULL_FRAME_BUFFER—Full frame buffer.
- R_GET_FRAME_BUFFER—Read frame buffer address.
- R_LINE_TYPE—Define line style and repeat length.
- R_LOCK_DEVICE—Lock device.
- R_OV_ECHO_COLORS—Select overlay echo colors.
- R_OVERLAY_ECHO—Select plane to contain cursor.
- R_TRANSPARENCY_INDEX—Specify HP 98720 transparency index.
- R_UNLOCK_DEVICE—Unlock device.
- READ_COLOR_MAP—Read color map.
- SWITCH_SEMAPHORE—Semaphore control.

Performance Tips

1. If only one process is accessing the graphics display, it is safe to turn off the semaphore operations. See the `SWITCH_SEMAPHORE gescape`. With semaphores turned off you can increase your program's speed 10 to 20 percent. If a tracking process is initiated, semaphores will automatically be turned on. See "Cautions" below for more information.
2. As with any driver, buffering is done to enhance performance. Performance can be degraded substantially if `buffer_mode` is turned off or an inordinate amount of `make_picture_current` calls are done.
3. Performance optimizations have been made so that sequential calls of the same output primitive with no intervening attribute changes or different primitive calls go faster. For example, the sequence `polygon, polygon, polyline, polyline` is faster then `polygon, polyline, polygon, polyline`. Also `line_color, polyline, polyline` is faster than `line_color, polyline, line_color, polyline`. So grouping by primitive and subgrouping primitives by attribute can give some performance improvements.
4. If Starbase echoes are in the overlay plane, graphics performance is significantly better since it is not necessary to "pick up" the cursor each time the frame buffer is updated.
5. Screen clears will be significantly faster if the area to be cleared starts on a 128-pixel boundary and is some multiple of 128-pixels wide. This can be checked by using the Starbase routines `transform_point` and `vdc_to_dc` to convert the bounds of the clear rectangle to device coordinates. Screen clears to the default `vdc_extent`will be aligned. Screen clears are also much faster when the background color index is zero. Screen clears with a non-zero index require two passes resulting in slower performance.
6. Polygons are filled faster when the drawing mode is ⟨*source*⟩, ⟨*not_source*⟩, `ZERO`, or `ONE`.
7. Horizontal and vertical lines are faster than diagonal lines on this device since the hardware block mover is used to generate the pixels.
8. The procedure `block_move` is faster than `block_read` or `block_write` since the hardware frame buffer block mover can be used.

9. Performance of `block_read` and `block_write` is significantly better if both the source and destination begin on the same byte boundary (since data can be transferred 32 bits at a time rather than one byte at a time). For example, one way to ensure this condition is to define pixel arrays as type short (16-bit integers) and then start `block_read` and `block_write` on even pixels only. This can more than double performance.

Cautions

The following cautions are provided in using this driver:

1. As mentioned previously, accessing the off-screen portion of the frame buffer (using `gescape`) should be done with care, since other processes access this region. See the section on offscreen usage for details.

2. Certain `gescape` functions should be used with caution since they bypass protection mechanisms used to prevent multiple processes from interfering with each other. For example, since the hardware resources can only be rationally used by one graphics process at a time, the driver activates a semaphore and locks the device before doing any output. This ensures, for example, that process A will not change the replacement rule while process B is in the middle of filling a polygon. It also prevents the terminal (`tty`) driver from overwriting any graphics processes that are outputting to the device. The driver unlocks the device when done processing output. Some of the `gescape` functions listed in the appendix for this manual allow the user to change this locking mechanism and should be used with *great caution.* Semaphores should never be turned off when operating in a window environment.

3. When using the HP 98720 device with a graphics accelerator it is possible for illegal operations to cause the transform engine or scan converter hardware to enter an unknown state. If this happens, Starbase will report an error the next time it tries to use the hardware. The user will see this as a `Transform engine timed out` or `Hardware/scan_converter time out` error. These are Starbase errors 14 and 52 respectively. This is a very serious error condition. If the HP 98721 device driver is being used, then this is a fatal error. When this error is discovered, Starbase reports the error and aborts execution.

   If an application needs to take some emergency action before an untimely termination, (such as saving valuable data) the application should check for these error conditions and take appropriate measures. Errors may be caught by an application using the `gerr_control` procedure described in the *Starbase Reference* manual.

It is also possible to avoid the termination completely if the application's error handler does not return control to Starbase. It is impossible, however, to proceed with any graphics efforts using the accelerator.

If the HP 98720 or HP 98720w drivers are being used to access the hardware and if they detect such an error; they will report the error condition, reset the transform engine, and continue (since they do not use the accelerator hardware).

Contents

HP98720W

The HP 98720w Window Device Driver

Device Description

The HP 98720 Graphics Display Station includes a high-resolution 19-inch color display, a display controller, and an optional graphics accelerator. See the table in the introduction for the SPUs that support this display.

Three device drivers are provided to access the HP 98720 Display:

- HP 98720—used to access graphics windows with the X Window system or the graphics display without using the optional graphics accelerator.
- HP 98720w—used to access graphics windows with HP Windows/9000, or the graphics display without windows. The latter is useful for doing fast alpha or font manager operations to the graphics display.
- HP 98721—used to access the graphics display using only the optional graphics accelerator.

This chapter covers the HP 98720w Device Driver. Refer to the "HP 98720 Device Driver" or "HP 98721 Device Driver" sections of this manual for information on using those device drivers.

Use the HP 98720w Device Driver to access HP Windows running on the HP 98720 Graphics Display Station (Series 300 only), or to use the fast alpha or font manager on the HP 98720 device.

This driver will *not* use the optional graphics accelerator, even if it is present. If you wish to use the graphics accelerator use the HP 98721 Device Driver. It also does *not* support X Windows; use the HP 98720 driver for that.

The display has a resolution of 1280×1024 pixels. The standard color display system has four-image planes. You can add optional memory in banks of eight-planes each. A fully configured system consists of three banks of image planes for full 24-bit per pixel color, one bank for full Z-buffer capability (with graphics

accelerator), and 4-overlay planes for non-destructive alpha, windows, cursors, or graphics.

If you add optional banks of frame buffer memory to the minimal system, the four standard image planes function as overlay planes. These overlay planes have their own unique color map, separate from the color map used for the newly installed image planes. This overlay color map consists of sixteen 4-bit entries. These four bits correspond to transparent, red, green, and blue (in order of Most Significant Bit (MSB) to Least Significant Bit (LSB). If the transparent bit (the MSB) is set to zero, the pixel color will be the color of the image 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 primary colors or transparent.

This driver, unlike the HP 98720 Device Driver and the HP 98721 Device Driver, supports only four planes in a combination of image or overlay planes. In the minimum system this driver provides full access to the four image planes allowing 16 colors to be displayed simultaneously from a pallet of 16 million. If your system has overlay planes, this driver uses only three overlay planes, which can display seven colors. These colors (indexes 0-7) are normally black, white, red, yellow, green, cyan, blue, and magenta, respectively. One of these colors must be reserved as **see-thru** to enable the user to see through objects in the overlay planes and into the frame buffer planes. By default index three, or yellow, is reserved as `see-thru`. This feature is discussed in more detail later. The HP 98720 and HP 98721 Device Drivers use the fourth plane to generate cursors.

The display system is a bit-mapped device with special hardware for:

- Video enable/disable individual image planes.
- Write enable/disable individual image and overlay planes.
- Memory writes with specified replacement rule (refer to `drawing_mode`).
- Video blinking of individual image planes.
- Arbitrary sized rectangular frame buffer to frame buffer copies.

The display is organized as an array of bytes, with each byte representing a pixel on the display. The four least significant bits of each byte determine the color, providing color map indexes from 0-15. The image plane color map is a RAM table that has 16 addressable locations and is 24 bits wide (8 bits each for

red, green, and blue). Thus, the pixel value in the image plane addresses the color map, generating the color programmed at that location. When four overlay planes are installed, the 3 least significant bits of each byte determine the color, providing color map indices from 0-7. The overlay plane color map is static.

Typically, the user does not need direct access to pixels in the frame buffer. However, for those applications which do require this, Starbase provides the gescape function R_GET_FRAME_BUFFER, which returns the virtual memory address of the beginning of the frame buffer. This gescape is discussed in the appendix of this manual. Frame buffer locations are then addressed relative to the returned address. The first byte of the frame buffer (byte 0) represents the upper left corner pixel of the screen. Byte 1 is immediately to its right. Byte 1279 is the last (right-most) pixel on the top line. The next 768 bytes of the frame buffer are not displayable. Byte 2048 is the first (left-most) pixel on the second line from the top. The last (lower right corner) pixel on the screen is byte number 2,096,383.

This off-screen portion of the frame buffer may be accessed via the gescape function R_FULL_FRAME_BUFFER documented in the appendix. Care should be taken when using this gescape since other processes, Starbase, and Windows/9000 access the frame buffer off-screen memory.

After reading this chapter, refer to the chapter "HP Windows/9000 Device Driver" to find out additional information on how this device driver is used with HP Windows/9000.

Series 800 On the Series 800 computers, a write to I/O space must be on the word boundaries. The frame buffer is mapped as integer (32 bits) per pixel. Therefore, when the user is writing directly to the frame buffer on the HP 98720 Graphics Display Station, each pixel is written with an integer access, and the pixel value is in the LSB of the integer value.

Usage and Restrictions

Windows/9000 and graphics applications that want to talk to a graphics window use this device driver. This driver does *not* support:

- bank switching
- Z-buffering
- double buffering
- shading
- the transform engine
- color map alterations in the overlay planes

HP Windows/9000 See-Thru Color

If you have installed optional frame buffer memory in the HP 98720 Graphics Display Station, Windows/9000 runs in the overlay planes. This allows the following:

- Applications can run in the image planes independent of Windows/9000.
- Applications using the transform engine in the image planes see no direct performance reduction by using Windows/9000 in the overlay planes.
- Windows/9000 can provide both opaque and transparent backgrounds.

The see-thru facility allows you to create a transparent window. Refer to the `windows`(1) information in the HP *Windows/9000 Reference* manual for details concerning this command.

By default index three, or yellow, is reserved as see-thru. The HP 98720w Device Driver recognizes a new Starbase environment variable `SB_OV_SEE_THRU_INDEX` that will allow the user to set the see-thru color map index to some other value. This environment variable will only have an effect when using Windows/9000 or when running the program on the `raw` device. Programs running in graphics windows use the `SB_OV_SEE_THRU_INDEX` value in effect at Windows/9000 startup time. Resetting this variable has no affect on programs using this driver. Any value outside the range 0-7 will be ignored. If this environment variable is not set or is set to an illegal value, the driver will default to using index three as see-thru.

Since the Term0 Server, Graphics Server, and Window Manager all use the HP 98720w driver to talk to this display, this environment variable also affects them. For example, the following situation will cause your Term0 text to seem to disappear:

1. a program sends color escape sequences to Term0 to get yellow-on-black characters
2. the see-thru index corresponds to what used to be yellow (3)
3. the image planes are cleared to black so that see-thru shows black

In another example, suppose your Windows/9000 environment variables are set up to have your window borders cyan-on-black. The borders will appear invisible if:

1. you set `SB_OV_SEE_THRU_INDEX=5` (cyan) before powering up Windows/9000
2. the image planes are cleared to black so that see-thru shows black

---

Setting Up the Device On a Series 300

The HP 98720w Device Driver can be used with the graphics display configured in either internal or external address space. Refer to the *Configuration Reference Manual* for a description of internal and external address space. If the device is configured as an external display, there will not be an Internal Terminal Emulator (ITE) for that device.

The Graphics Interface Card may be installed in any DIO slot in the computer's backplane or in any I/O slot of the expander.

Switch Settings

The Graphics Interface Card has a single 6-bit address select switch. One bit, labelled `FB`, determines the frame buffer location, while the other five switch bits, labelled `CS`, determine the location within the DIO memory map of the HP 98720 control space. Silkscreening on the printed circuit board indicates the meaning of the bits.

The frame buffer consumes two Mbytes of I/O address space starting at FB_BASE. The switch bit labelled FC determines the address of FB_BASE as shown below.

Table HP98720W-1. Frame Buffer Locations

FB	FB_BASE (hex)
0	$200000
1	$800000

Typical systems will map the frame buffer to $200000. However, some systems which have multiple displays may map the frame buffer address to $800000. When the frame buffer address is set to $800000, the HP Series 300 Model 320 SPU memory limit is reduced from 7.50 Mbytes to 5.75 Mbytes. This occurs since the frame buffer is mapped into the upper 2 Mbytes of memory address space.

The control space requires 128 Kbytes starting at "CTL_BASE". The five switch bits labelled CS determine the address of "CTL_BASE". The HP 98720 may be configured as an external or internal display. Since only 64 Kbytes are normal allotted for external I/O select codes, two consecutive select codes will be consumed if the device is configured as an external device. The control space may be located at any of 32 positions. Sixteen positions are reserved in internal I/O space and sixteen are in external I/O space (with 5 reserved). The table below lists the binary switch settings with the corresponding values of CTL_BASE for external I/O settings, as well as the select code spaces consumed.

Table HP98720W-2. Control Space Settings (External I/O)

CS Setting	CTL_BASE (hex)	Select Codes
01011	$560000	
10101	$6A0000	10-11
10110	$6C0000	12-13
10111	$6E0000	14-15
11000	$700000	16-17
11001	$720000	18-19
11010	$740000	20-21
11011	$760000	22-23
11100	$780000	24-25
11101	$7A0000	26-27
11110	$7C0000	28-29
11111	$7E0000	30-31

If the HP 98720 is configured as the system console, CTL_BASE needs to be placed at $560000, which is an internal I/O setting. If the device is *not* used as the system console then the control space should *not* be placed in internal I/O space, since it is likely to overlap the address space of other system hardware. In this case, an external I/O space setting should be selected with two consecutive select codes which are unused by the system.

Note This display driver does not know how to reset the device's hardware when the control space is configured for an external display. Hence, you must first do a **gopen** to the device with a mode of RESET_DEVICE using either the HP 98720 or HP 98721 driver before using the HP 98720w driver or running windows on this device. A small and simple program called from **/etc/rc** during power up which resets the device is a good solution. As example program that accomplishes this reset follows.

Example Program

(To reset HP 98720 on Series 300 external select code)

```
/*
 * Starbase program: reset98720.c
 * Compile: cc -o reset98720 reset98720.c -ldd98720 -lsb1 -lsb2
 * Destination: /usr/bin
 * Execute: add line to the /etc/rc - "/usr/bin/reset98720 /dev/crt.external"
 *
 * Example program to be put in /etc/rc for Resetting an external HP 98720
 * device during power up. This is done so that the window system or any
 * other application that uses the HP 98720w device driver can be run to
 * the HP 98720 display. This is necessary because the HP 98720w does not
 * know how to reset the external device.
 */
#include <starbase.c.h>

main(argc,argv)
int argc; char *argv[];
{
   int fildes;

   if ((fildes = gopen(argv[1],OUTDEV,"hp98720",INIT|RESET_DEVICE)) < 0)
      printf("External HP 98720 %s initialization failed.\n",argv[1]);
   else {
      printf("External HP 98720 %s initialization succeeded.\n",argv[1]);
      gclose(fildes);
   }
}
```

Setting Up the Device On Series 800

Up to four HP 98720 devices can be connected to most supported Series 800 SPUs, but it is recommended that only two of these have an Internal Terminal Emulator (ITE) running on them.

Only one HP 98720 device can be connected to a Series 800 Model 840.

Special Device Files (mknod)

See the *Windows/9000 Reference* for details on how special files are created for windows.

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 details. 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. These files 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 `crt`.

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

For Series 300

When the device is at the internal address (refer to the "Switch Settings" segment of this section) the `mknod` parameters would create a character special device with a major number of 12 and a minor number of 0.

```
mknod /dev/crt c 12 0x00000
```

When the device is at an external address (refer to the "Switch Settings" segment of this section) the `mknod` parameters would create a character special device with a major number of 12 and a minor number of `0x`⟨*sc*⟩`0200` where ⟨*sc*⟩ is the two-digit external select code. Note that the leading `0x` causes the number to be interpreted hexadecimally.

```
mknod /dev/crt c 12 0x⟨sc⟩0200
```

For Series 800

When creating the device file, use `mknod` with a major number of 14 and a minor number as indicated below (where ⟨*lu*⟩ is two-digit the hardware logical unit number).

```
mknod /dev/crt c 14 0x00⟨lu⟩00
<<or>>
mknod /dev/crtx c 14 0x00⟨lu⟩00
```

Linking the Driver

The HP 98720w Device Driver is the file `libdd98720w.a` in the `/usr/lib` directory. This device driver may be linked to a program using the absolute path name `/usr/lib/libdd98720w.a`, an appropriate relative path name, or by using the `-l` option `-ldd98720w`. For example, to compile and link a program with this driver use:

```
cc example.c -ldd98720w -lsb1 -lsb2 -o example
fc example.f -ldd98720w -lsb1 -lsb2 -o example
pc example.p -ldd98720w -lsb1 -lsb2 -o example
```

depending upon the language being used.

Device 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 Windows/9000 as specified in the last section (for example, `/dev/screen/crt`).

Kind — Indicates the I/O characteristics of the device. This parameter may be one of the following:

- `INDEV`—Input only

- OUTDEV—Output only
- OUTINDEV—Input and Output

Driver — The character representation of the driver type. This must be hp98720w. For example:

`"hp98720w"`	*for C.*
`'hp98720w'//char(0)`	*for FORTRAN77.*
`'hp98720w'`	*for Pascal.*

Mode — The mode control word consisting of several flag bits *or* ed together. Listed below are those flag bits having device-dependent actions. Those flags not discussed below operate as defined by the gopen procedure. This control word has no effect on the color map when using the overlay plane configuration; the software color map is always initialized to the appropriate values.

- SPOOLED—cannot spool raster devices.
- MODEL_XFORM—Shading is *not* supported for this device.
- 0—open the device, but do nothing else. The software color map is initialized from the current hardware color map.
- INIT—open and initialize the device as followsr:
  1. Clear image planes, overlay planes, or graphics window to 0s.
  2. Reset the color map to its default values, if using the image plane configuration.
  3. Enable the display for reading and writing.

Syntax Examples

To open and initialize an HP 98720w device for output:

For C Programs:

```
fildes = gopen("/dev/crt",OUTDEV,"hp98720w",INIT);
```

For FORTRAN77 Programs:

```
fildes = gopen('/dev/crt'//char(0),OUTDEV,'hp98720w'//char(0),INIT)
```

For Pascal Programs:

```
fildes = gopen('/dev/crt',OUTDEV,'hp98720w',INIT);
```

Special Device Characteristics

For device coordinate operations, location $(0, 0)$ is the upper-left corner of the screen with X-axis values increasing to the right and Y-axis values increasing down. The lower-right corner of the display is $(1279, 1023)$.

Device Defaults

Number of Color Planes

When the gopen procedure is called, this driver asks the device for the number of color planes available. This number can be either 3 or 4 depending on the hardware configuration. The device driver then acts accordingly.

Dither Default

The default number of colors searched for in a dither cell is 2. The number of colors allowed in a dither cell is 1, 2, 4, 8 or 16.

Raster Echo Default

The default raster echo is the 8×8 array:

15	15	15	15	0	0	0	0
15	15	0	0	0	0	0	0
15	0	15	0	0	0	0	0
15	0	0	15	0	0	0	0
0	0	0	0	15	0	0	0
0	0	0	0	0	15	0	0
0	0	0	0	0	0	15	0
0	0	0	0	0	0	0	15

The maximum size allowed for a raster echo is 64×64 pixels. The default drawing mode for the raster echo is 7 (a logical *OR*).

Color Plane Defaults

The minimum configuration is a color mapped system with 4-image planes. Extended configurations are static color mapped systems with 3-overlay planes.

The latter can occur regardless of the number of image plane banks installed. All planes being used are display enabled and write enabled.

Semaphore Default

Semaphore operations are enabled.

Line Type Defaults

The default line types are created with the following bit patterns:

Table HP98720W-3.

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

Default Color Map

If the fourth gopen parameter is zero (0), the current hardware color map is used on color displays. Indexes 0-7 of the table shown below are the values assigned to the software color map configured with overlay planes. One entry must be set to see-thru, either by default or by setting the `SB_OV_SEE_THRU_INDEX` environment variable.

If the fourth gopen parameter is `INIT`, the current color map is initialized to the default values shown below. Again, all but one of the 0-7 indexes apply in the overlay plane configuration.

Table HP98720W-4. 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

Use the `inquire_color_table` procedure to see the rest of the available colors.

Red, Green, and Blue

Each file descriptor opened as an output device has a color table associated with it. If multiple file descriptors are open to the same device, the color table and the device's color map may not always be identical. The color table does not track the color map if the device's color map is changed by another file descriptor path.

Starbase Functionality

Commands Not Supported

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

Commands Conditionally Supported

The following commands are supported under the listed conditions:

`block_read`, `block_write`	The "`raw`" parameter for the `block_read` and `block_write` commands is normally ignored by this device driver. To use the "`raw`" mode, you must call the `gescape` function `R_BIT_MODE`. The Appendix A discusses the `gescape` functions common to several drivers.
`define_color_table`	When running in the overlay planes, this command has no effect.
`display_enable`	When running in the overlay planes, this command has no effect.
`inquire_color_table`	When running in the overlay planes, this command returns the software color map values.
`pattern_define`	4×4 is the largest pattern supported on this device.
`text_precision`	Only `STROKE_TEXT` precision is supported.

`vertex_format`	The `use` parameter must be zero. Any extra coordinates supplied will be ignored.

Fast Alpha and Font Manager Functionality

This device driver supports raster text calls from the fast alpha and font manager libraries. See the *Fast Alpha/Font Manager's Programmer's Manual* for further information.

Parameters for gescape

The following `gescape` functions are common to many of the Hewlett-Packard displays supported by Starbase. Detailed information about these functions can be found in Appendix A.

- `SWITCH_SEMAPHORE`—semaphore control
- `READ_COLOR_MAP`—read color map
- `BLINK_DISPLAY`—blink display (Blink rate is 3.75 Hz for this device and only when `gopened` as a 4-plane device.)
- `R_GET_FRAME_BUFFER`—read frame buffer address
- `R_FULL_FRAME_BUFFER`—full frame buffer
- `R_LOCK_DEVICE`—lock device
- `R_UNLOCK_DEVICE`—unlock device
- `R_BIT_MODE`—bit mode
- `R_BIT_MASK`—bit mask
- `R_DEF_FILL_PAT`—define fill pattern

Performance Tips

1. Horizontal and vertical lines are faster than diagonal lines on these devices since the hardware block mover is used to generate the pixels.
2. The procedure `block_move` is faster than `block_read` or `block_write` since the hardware frame buffer block mover can be used.
3. As with any driver, buffering is done to enhance performance. If `buffer_mode` is turned off or if you make many calls to `make_picture_current` then performance can decrease.
4. Performance optimizations have been made so that sequential calls of the same output primitive with no intervening attribute changes or different primitive calls go faster. For example the sequence `polygon, polygon, polyline, polyline` is faster the `polygon, polyline, polygon, polyline`. Also `line_color, polyline, polyline` is faster than `line_color, polyline, line_color, polyline`. So grouping by primitive and subgrouping primitives by attribute can give some performance improvements.

Cautions

1. Use care when accessing the off-screen portion of the frame buffer (via `gescape` functions), since other processes access this region. The first 32 lines of off-screen frame buffer memory are used by Windows/9000 for its sprite and by this device driver for polygon fills. The rest of the off-screen frame buffer memory can be consumed by font bit maps when Windows/9000 is running.
2. Certain `gescape` functions should be used with caution since they bypass protection mechanisms used to prevent multiple processes from interfering with each other. For example, only one graphics process should access a hardware resource at any time. To enforce this the driver activates a semaphore and locks the device before doing any output. This ensures, for example, that process A will not change the replacement rule while

process B is in the middle of filling a polygon. It also prevents the terminal (`tty`) driver from overwriting any graphics processes that are outputting to the device. The driver unlocks the device when it is finished processing output. Some of the `gescape` functions listed in this section allow the user to change this locking mechanism and should be used with *great caution.*

3. When using an HP 98720 device having a graphics accelerator, it is possible for illegal operations to cause the transform engine or scan converter hardware to enter an unknown state. If this happens, Starbase will report an error the next time it tries to use the hardware. The user will see this as a `Transform engine timed out` or `Hardware/scan_converter time out` error. These are Starbase errors 14 and 52 respectively. This is a very serious error condition. If the HP 98721 Device Driver is being used, this is a fatal error. When this error is discovered, Starbase reports the error and aborts execution.

   If an application needs to take some emergency action before an untimely termination (such as saving valuable data), the application should check for these error conditions and take appropriate measures. Errors may be caught by an application using the `gerr_control` procedure described in the *Starbase Reference* manual.

   It is also possible to avoid the termination completely if the application's error handler does not return control to Starbase. It is impossible to proceed, however, with any graphics efforts using the accelerator.

   If the HP 98720 or HP 98720w drivers are being used to access the hardware and such an error is detected, they will report the error condition, reset the transform engine, and continue (since they do not use the accelerator hardware).

Contents

HP98721

The HP 98721 Device Driver

Device Description

The HP 98721A is an optional graphics accelerator for the HP 98720A controller. The controller plugs into an I/O slot of the SPUs. (See the "Introduction" section of this manual for systems supporting this controller and accelerator.)

Three device drivers are provided to access the HP 98720 display:

- HP 98720—used to access graphics windows in the X Window system or the graphics display without using the optional graphics accelerator.
- HP98720w—used to access graphics windows with HP Windows/9000, or the graphics display without windows. The latter is useful for doing fast alpha or font manager operation to the display.
- HP 98721—used to access the graphics display using only the optional graphics accelerator.

This chapter covers the HP 98721 Device Driver; see the "HP 98720 Device Driver" or "HP 98720w Device Driver" chapters for information on using those device drivers.

The display has a resolution of 1280×1024 pixels. The standard color display system has four planes of frame buffer to provide 16 simultaneous colors. You can add optional memory in banks of eight planes each. A fully configured system consists of three banks of frame buffer for full 24 bit per pixel color, one bank for full Z-buffer capability, and 4-overlay planes for non-destructive alpha, cursors, or graphics. In order to use the HP 98721 Device Driver, the system must be configured with a graphics accelerator and at least one bank of eight planes. Four-plane systems are not supported with the graphics accelerator.

An 8-plane configuration allows 256 colors to be displayed simultaneously from a palette of 16 million. A 16-plane system is like two 8-plane frame buffers where only one 8-plane buffer is displayed at any time. This configuration is useful for

double buffering. When three banks of frame buffer are installed, the system may be configured to display eight bits red, eight bits green, and eight bits blue per pixel. Double buffering may also be achieved at a resolution of four bits red, four bits green, and four bits blue.

The display system is a bit-mapped device with special hardware for:

- Write enable/disable individual planes.
- Video enable/disable individual planes.
- Memory writes with specified replacement rule (see `drawing_mode` in *Starbase Reference* manual).
- Video blinking of individual planes.
- Video blinking of individual color map locations.
- Arbitrary sized rectangular memory to memory copies.
- Write enable/disable of pixels in 4×4 cell for "screen door" transparency.
- Bit-slice processor with hardware floating point for high speed three-dimensional transformations.
- NMOS III scan converter with six axis interpolation for Gouraud shaded, Z-buffered vectors and polygons.

The display is organized as an array of bytes, with each byte representing a pixel on the display. With four planes installed, the 4 LSBs of each byte determine the color providing color map indices from 0–15 (this is not a supported configuration). When eight planes are installed, color map indices range from 0–255. The color map is a RAM table that has 16 or 256 addressable locations and is 24 bits wide (eight bits each for red, green, and blue). Thus, the pixel value in the frame buffer addresses the color map, generating the color programmed at that location.

If you add optional banks of frame buffer memory to the minimal system, the four standard image planes function as overlay planes. These overlay planes have their own unique color map, separate from the color map used for the newly installed image planes. This overlay color map consists of sixteen 4-bit entries. These four bits correspond to transparent, red, green, and blue (in order of MSB ot LSB.) If the transparent bit (the MSB) is set to zero, the pixel color will be the color of the image 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 primary colors or transparent.

You can use overlay planes for non-destructive alpha, graphics, or cursors. For example, when the HP 98720 is used as system console, the Internal Terminal Emulator (ITE) uses three of the overlay planes for alpha information. This way there is no interaction between ITE text and images in the graphics planes. Windows/9000 also runs in the overlay planes (refer to the "HP 98720w Device Driver" chapter for more information). To do graphics in the overlay planes the HP 98721 Device Driver may be opened directly to the overlay planes as if they were a separate device (refer to the section "Setting up the Device" for more information). One overlay plane is reserved for graphic cursors.

When Starbase cursors are in the overlay plane performance is enhanced since it it not necessary to *pick up* the cursor each time the frame buffer is updated. You can think of the overlay plane used for cursors as a separate "cursor plane." Any data in the cursor plane will be displayed *over* data in the graphics planes. Data in the other three overlay planes will be displayed over data in the graphics planes and the cursor plane. For example, suppose a graphics application is running in the graphics planes while the window manager is running. If the application has a Starbase cursor in the overlay cursor plane, the cursor will always be visible inside regions of see-thru because the cursor has display priority over the graphics. (Refer to the "HP Windows/9000 See-Thru Color" section of the "HP 98720w Device Driver".) If the cursor is moved outside the graphics window boundary it is not visible since the window desktop environment is drawn to the overlay planes (which have display priority over the cursor plane).

Typically, the user does not need to directly read or write pixels in the frame buffer. However, for those applications which require direct access, Starbase does provide the `gescape` procedure `R_GET_FRAME_BUFFER`, which returns the virtual memory address of the beginning of the frame buffer. This gescape is discussed in the appendix of this manual. Frame buffer locations are then addressed relative to the returned address. The first byte of the frame buffer (byte 0) represents the upper left corner pixel of the screen. Byte 1 is immediately to its right. Byte 1279 is the last (right-most) pixel on the top line. The next 768 bytes of the frame buffer are not displayable. Byte 2048 is the first (left-most) pixel on the second line from the top. The last (lower right corner) pixel on the screen is byte number 2,096,383.

If more than one optional frame buffer bank is installed, bank switching must be used to access the additional memory. A number of Starbase calls may set the bank register so it is advisable to call `bank_switch` just prior to making accesses to the frame buffer pointer to ensure desired results.

The off-screen portion of the frame buffer may be accessed via the `gescape` procedure `R_FULL_FRAME_BUFFER` documented in the appendix of this manual. Use this `gescape` carefully since other processes, Starbase, X Windows, and HP Windows/9000 access the frame buffer off-screen memory. This driver is not supported running concurrently in the same planes as X Windows or HP Windows/9000.

Series 800

On the Series 800 computers, a write to IO space must be on the word boundaries. The frame buffer is mapped as integer (32bits) per pixel. Therefore, when the user is writing directly to the frame buffer on the HP 98720 Graphics Display Station, each pixel is written with an integer access.

If writing to the HP 98720 image buffer and not in `CMAP_FULL` color map mode, only one bank can be written at a time. The bank to be written must be established by a call to `bank_switch`. The following conditions exist at the time of writing the pixel value:

- When writing to bank 0, the pixel value is in the LSB of the integer value.
- When writing to bank 1, the pixel value is in byte position 1 of the integer value (where the LSB is byte position 0).
- When writing to bank 2, the pixel value is in byte position 2 of the integer value.

All three banks for one pixel can be written simultaneously by packing all three bank values for the pixel into the integer value and having the color map mode as `CMAP_FULL` before writing.

Setting Up the Device On Series 300

The HP 98721 Device Driver can be used with the graphics display configured in either internal or external address space (refer to the *Configuration Reference Manual* for a description of internal and external address space.)

Note If the HP 98720 is configured as an external display, there will *not* be an Internal Terminal Emulator (ITE) for that device. Since it is the ITE that normally initializes the display. External devices must be reset after power-up by running a simple Starbase program with a mode of `RESET_DEVICE` in the `gopen` call. It may also be necessary to run this program after running an application which manipulated the overlay color map, such as a windows application program. An example program which could be called from `/etc/rc` during power-up is given at the end of this section. For more details concerning the effects of `RESET_DEVICE`, see the "Device Initialization" information in this section.

The Graphics Interface card may be installed in any DIO slot in the computer's backplane or in any I/O slot of the expander.

Switch Settings

The Graphics Interface Card has a single 6-bit address select switch. One bit, labeled `FB`, determines the frame buffer location, while the other five switch bits, labeled `CS`, determine the location within the DIO memory map of the HP 98720 control space. Silkscreening on the printed circuit board indicates the meaning of the bits.

The frame buffer consumes 2 Mbytes of I/O address space, starting at `FB_BASE`. The switch bit labeled `FB` determines the address of `FB_BASE` as shown below.

Table HP98721-1. Frame Buffer Locations

FB	FB_BASE (hex)
0	$200000
1	$800000

Typical systems will map the frame buffer to $200000. However, some systems which have multiple displays may map the frame buffer address to $800000. When the frame buffer address is set to $800000, the Hewlett-Packard Series 300 Model 320 SPU memory limit is reduced from 7.50 Mbytes to 5.75 Mbytes. This occurs since the frame buffer is mapped into the upper 2 Mbytes of memory address space.

The control space requires 128 Kbytes starting at CTL_BASE. The five switch bits labelled CS determine the address of CTL_BASE. The HP 98720 may be configured as an external or internal display. Since only 64 Kbytes are normally allotted for external I/O select codes, two consecutive select codes will be consumed if the device is configured as an external display. The control space may be located at any of 32 positions. In internal I/O space 16 positions are reserved. There are 16 in external I/O space with 5 reserved. The table below lists the binary switch settings with the corresponding values of CTL_BASE for external I/O settings, as well as the select code spaces consumed.

Table HP98721-2. Control Space Settings (External I/O)

CS Setting	CTL_BASE (hex)	Select Codes
01011	$560000	
10101	$6A0000	10-11
10110	$6C0000	12-13
10111	$6E0000	14-15
11000	$700000	16-17
11001	$720000	18-19
11010	$740000	20-21
11011	$760000	22-23
11100	$780000	24-25
11101	$7A0000	26-27
11110	$7C0000	28-29
11111	$7E0000	30-31

If the HP 98720 is configured as the system console, CTL_BASE needs to be placed at $560000, which is an internal I/O setting. If the device is not used as the system console, the control space should not be placed in internal I/O space since it is likely to overlap the address space of other system hardware. In this case, an external I/O space setting should be selected with two consecutive select codes which are unused by the system.

Example Program to Reset the HP 98720

```
/*
 * Starbase program: reset98720.c
 * Compile: cc -o reset98720 reset98720.c -ldd98720 -lsb1 -lsb2
 * Destination: /usr/bin
 * Execute: add line to the /etc/rc - "/usr/bin/reset98720 /dev/crt.external"
 *
 * Example program to be put in /etc/rc for resetting an external HP 98720
 *  device during power-up.
 */
#include <starbase.c.h>

main(argc,argv)
int argc; char *argv[];
{
   int fildes;

   if ((fildes = gopen(argv[1],OUTDEV,"hp98720",INIT|RESET_DEVICE)) < 0)
      printf("External HP 98720 %s initialization failed.\n",argv[1]);
   else {
      printf("External HP 98720 %s initialization succeeded.\n",argv[1]);
      gclose(fildes);
   }
}
```

Setting Up the Device On Series 800

Up to four HP 98721 devices can be connected to most supported Series 800 SPUs. However, it is recommended that only two of these have an Internal Terminal Emulator or window system running on them.

Only one HP 98721 device can be connected to a Series 800 Model 840.

Special Device Files (mknod) On Series 300

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 `crt`.

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

When the device is at the internal address (refer to the "Switch settings" section) the `mknod` parameters should create a character special device with a major number of 12 and a minor number of 0. Note that the leading `0x` causes the number to be interpreted hexadecimally.

```
mknod /dev/crt c 12 0x000000
```

When the device is at an external address (refer to the "Switch settings" section) the `mknod` parameters should create a character special device with a major number of 12 and a minor number of `0x`⟨*sc*⟩`0200` where ⟨*sc*⟩ is the two-digit external select code.

```
mknod /dev/crt c 12 0x⟨sc⟩0200
```

The HP 98721 Device Driver may also be used for the overlay planes in graphics mode (if they are present.) Since one plane is still reserved for cursors, the graphics device will look like a 3-plane device in this mode. Note that since the terminal emulator and window system operate in the overlay planes also, there will be interactions with these processes if a graphics driver is opened in this manner while these processes are present. To open the HP 98721 Device Driver to the overlay planes instead of the graphics planes, the last byte of the minor number must be 1.

For example, when the device is at an internal address, the `mknod` parameters for the overlay device should create a character special device with a major number of 12 and a minor number of 1.

```
mknod /dev/ocrt c 12 0x000001
```

When the device is at an external address (refer to the section on "Switch Settings") the `mknod` parameters for the overlay device should create a character special device with a major number of 12 and a minor number of `0x`⟨*sc*⟩`0201` where ⟨*sc*⟩ is the two-digit external select code.

```
mknod /dev/ocrt c 12 0x⟨sc⟩0201
```

Special Device Files (mknod) On Series 800

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 *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 names that are suggested for these devices are `crt`, `crt0`, `crt1`, or `crt2`.

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

When creating the device file, the `mknod` parameters should create a character special device with a major number of 14 and a minor number of the format below (where ⟨*lu*⟩ is the two-digit hardware logical unit number):

```
mknod /dev/crtx c 14 0x00⟨lu⟩00
```

The HP 98721 Device Driver may also be used for the overlay planes in graphics mode (if they are present.) Since one plane is still reserved for cursors, the graphics device will look like a 3-plane device in this mode. Note that since the

terminal emulator and window system operate in the overlay planes also, there will be interactions with these processes if a graphics driver is opened in this manner while these processes are present. To open the HP 98721 Device Driver to the overlay planes instead of the graphics planes, the last byte of the minor number must be 1.

For example, the `mknod` parameters for the overlay device should create a character special device with a major number of 14 and a minor number of the format indicated below (where ⟨*lu*⟩ is the two-digit hardware logical unit number):

```
mknod /dev/ocrtx c 14 0x00⟨lu⟩01
```

Linking the Driver

The HP 98721 Device Driver is the file named `libdd98721.a` in the `/usr/lib` directory. This device driver may be linked to a program using the absolute path name `/usr/lib/libdd98721.a`, an appropriate relative path name, or by using the `-l` option `-ldd98721`. For example, to compile and link a program for use with this driver, use:

```
cc example.c -ldd98721 -lsb1 -lsb2 -o example
fc example.f -ldd98721 -lsb1 -lsb2 -o example
pc example.p -ldd98721 -lsb1 -lsb2 -o example
```

depending upon the language being used.

Device Initialization

Parameters for gopen

Note Because the transform engine is not multi-tasking, only *one* HP 98721 driver may be opened to a device file. Other HP 98720 drivers may be opened to that device file if multiple Starbase drivers are needed.

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 (for example, `/dev/crt`.)

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 `hp98721`. For example:

`"hp98721"`	*for C.*
`'hp98721'//char(0)`	*for FORTRAN77.*
`'hp98721'`	*for Pascal.*

Mode — The mode control word consisting of several flag bits which are *or* ed together. Listed below are the flag bits which which have device-dependent actions. Those flags not discussed below operate as defined by the gopen procedure.

- `SPOOLED`—cannot spool raster devices.
- `0`—open the device, but do nothing else. The software color map is initialized on monochrome monitors.
- `INIT`—open and initialize the device as follows:
  1. Clear frame buffer to 0s.
  2. Reset the color map to its default values.
  3. Enable the display for reading and writing.
  4. Download the transform engine's microcode.
- `RESET_DEVICE`—open and reset the device as follows:

1. Clear frame buffer and overlays to 0s.
2. Reset the color map to its default values.
3. Clear the overlay color map.
4. Enable the display for reading and writing.
5. Download the transform engine's microcode.

Note that the RESET_DEVICE flag bit should be used with caution: it will adversely affect any other processes using the device. This flag bit is intended to reset a device completely. This should only be necessary for devices in an unknown state such as a device powered up in an external I/O space. Most programs should not use this flag bit.

Syntax Examples

To open and initialize an HP 98721 device for output:

For C Programs:

```
fildes = gopen("/dev/crt",OUTDEV,"hp98721",INIT);
```

For FORTRAN77 Programs:

```
fildes = gopen('/dev/crt'//char(0),OUTDEV,'hp98721'//char(0),INIT)
```

For Pascal Programs:

```
fildes = gopen('/dev/crt',OUTDEV,'hp98721',INIT);
```

Special Device Characteristics

For device coordinate operations, location $(0,0)$ is the upper-left corner of the screen with X-axis values increasing to the right and Y-axis values increasing down. The lower-right corner of the display is therefore $(1279, 1023)$.

Device Defaults

Number of Color Planes

When the gopen procedure is called, this driver asks the device for the number of color planes available. This number can be either 3(in overlays), 8, 16, or 24. The number 4 is not a supported configuration. The device driver then acts accordingly.

Dither Default

The default number of colors searched for in a dither cell is 2. The number of colors allowed in a dither cell is 1, 2, 4, 8 or 16.

Raster Echo Default

The default raster echo is the 8×8 array:

255	255	255	255	0	0	0	0
255	255	0	0	0	0	0	0
255	0	255	0	0	0	0	0
255	0	0	255	0	0	0	0
0	0	0	0	255	0	0	0
0	0	0	0	0	255	0	0
0	0	0	0	0	0	255	0
0	0	0	0	0	0	0	255

The maximum size allowed for a raster echo is 64×64 pixels. The default drawing mode for the raster echo is 7 (a logical *OR*).

By default the raster echo is written to the graphics planes. All other echo types are written to an overlay plane. the location of raster and non-raster echoes can be changed using the `R_OVERLAY_ECHO gescape`.

Color Planes Defaults

The default configuration is an 8-plane color mapped system regardless of the number of frame buffer banks installed.

All planes in the first bank are display enabled. All planes in the first bank are write enabled.

Semaphore Default

Semaphore operations are enabled.

Line Type Defaults

The default line types are created with the bit patterns shown below:

Table HP98721-3.

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

Default Color Map

If the fourth gopen parameter is zero (0) then the current hardware color map is used on color displays.

If the fourth gopen parameter is INIT, then the current color map is initialized to the default values shown below.

Table HP98721-4. 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

Use the inquire_color_table procedure to see the rest of the 255 colors.

When INIT is used in the shade_mode procedure call the color map initialization is based on the value of the mode parameter and on the number of frame buffer banks installed.

mode=CMAP_NORMAL — Same as the previous table above. Only one bank of the first two can be displayed at a time. If a third bank is installed it can not be displayed in this mode.

`mode=CMAP_MONOTONIC`

The color map will be initialized as:

```
for (i=0; i<256; i++) {
  cmap[i].red = cmap[i].
  green = cmap[i].blue = i/255.0;
}
```

Only one bank of the first two can be displayed at a time. If a third bank is installed it can not be displayed in this mode.

`mode=CMAP_FULL`

With less than three banks installed the color map will be initialized as three bits red, three bits green, and two bits blue. The three most significant bits are red and the two least significant bits are blue. Only one bank of the first two can be displayed at a time.

Note: this driver requires at least eight planes in `CMAP_FULL` mode, or at least 16 planes if double buffered.

With three or more banks installed the color map will be initialized as the `CMAP_MONOTONIC` case above, but now the first bank of eight will go through the blue portion of the color map, the second bank goes through the green portion, and the third bank goes through the red portion. In this mode the color map is transparent and the eight bits from each bank drive the appropriate DAC. The color map could be subsequently modified in this mode to perform functions such as gamma correction or double buffering of 4 bits per color.

Red, Green and Blue

Each file descriptor opened as an output device has a color table associated with it. If multiple file descriptors are open to the same device, the color table and the device's color map may not always be identical. The color table does not track the color map if the device's color map is changed by another file descriptor path.

For Starbase procedures that have parameters for red, green and blue, the way the actual color is chosen depends on the current **shade_mode** setting.

`mode=CMAP_NORMAL`

The color map is searched for the color that is closest in RGB space to the one requested. That color map index is written to the frame buffer for subsequent output primitives. It is more efficient to select a color with an index rather than specifying a color with red, blue, and green values in this mode because it takes extra time to figure out which index in the color table most closely matches the specified color.

`mode=CMAP_MONOTONIC`

The red, green, and blue value is converted to an intensity value using the equation:

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

This intensity is converted to an index value by mapping intensity 0.0 to the minimum index set be **shade_range** and intensity 1.0 to the maximum index set by **shade_range**. This mode is useful for displaying a high quality monochrome picture on an 8-plane system from data that produces a high quality color picture on a 24-plane system.

`mode=CMAP_FULL`

With less than three banks installed, the color is converted to a color map index by the equation:

```
index=(round(red*32767)>>7) & 0xE0 |
(round(green*32767)>>10) & 0x1C |
(round(blue*32767)>>13)
```

This equation will be used in this mode regardless of whether the user has modified the color map.

With three or more banks installed, the red, green and blue values are each multiplied by 32,767, converted to an integer, shifted right seven places, then written to the appropriate bank.

Starbase Functionality

Commands Not Supported

These procedures are ignored.

`bf_control`	`bf_surface_model`
`bf_fill_color`	`depth_cue_color`
`bf_interior_style`	`depth_cue_range`
`bf_perimeter_color`	`interior_style (INT_OUTLINE)`
`bf_perimeter_repeat_length`	`interior_style (INT_POINT)`
`bf_perimeter_type`	`light_attenuation`
`bf_surface_coefficients`	`surface_coefficients`

Commands Conditionally Supported

The following commands are supported under the listed conditions:

`backface_control` — Backface color is supported only if shading is on as set by `shade_mode`. Also, `backface_color` does not work correctly for spline surfaces when `VERTEX_FORMAT` specifies normal per vertex.

`interior_style` — If the polygon fill is `INT_HATCH`, the following functionality will not work correctly.

- Hidden surface removal.
- Shading and lighting.
- Depth cueing.
- Backfacing attributes and culling.
- Splines, quadralateral meshes, and triangle strips, will not be hatched.

Performance is also degraded in this mode.

`light_model`
`light_source`
`light_switch` — Supports nine light sources; one ambient and eight positional or directional.

`pattern_define` — 4×4 is the largest pattern supported on this device.

`text_precision` — Only `STROKE_TEXT` precision is supported.

`vertex_format` The *use* parameter cannot be greater than three(3); no more than three extra coordinates per vertex can be interpreted by this device. Extra values can only be rgb triplets, vertex normals, or vertex intensity values. Values of *use* greater than three will cause the extra data per vertex to be ignored.

block_read, block_write

The "raw" parameter for the `block_read` and `block_write` commands is normally ignored by this device driver. To use the `raw` mode, you must call the `R_BIT_MODE gescape` discussed in the appendix of this manual. If the "RAW" parameter is `TRUE`, no clipping will be done.

Splines

`QUARTIC` and `QUINTIC` splines (that is, fifth and sixth order splines) are not supported on this driver.

The commands that are affected are `spline_curve2d`, `spline_curve3d`, and `spline_surface`.

Fast Alpha and Font Manager

This device driver does *not* support raster text calls from the fast alpha and font manager library.

Parameters for gescape

The following gescape functions are common to two or more of the Hewlett-Packard displays supported by Starbase. Detailed information about these functions can be found in Appendix A.

- BLINK_INDEX—Alternate between HP 98720 hardware color maps.
- BLINK_PLANES—Blink display (blink rate is 3.75 Hz for this device).
- LS_OVERFLOW_CONTROL—Sets options for overflow situations.
- PATTERN_FILL—Fill polygon with stored pattern.
- R_BIT_MASK—Bit mask.
- R_BIT_MODE—Bit mode.
- R_DEF_ECHO_TRANS—Define raster echo transparency.
- R_DEF_FILL_PAT—Define fill pattern.
- R_FULL_FRAME_BUFFER—Full frame buffer.
- R_GET_FRAME_BUFFER—Read frame buffer address.
- R_LINE_TYPE—Define line style and repeat length.
- R_LOCK_DEVICE—Lock device.
- R_OV_ECHO_COLORS—Select overlay echo colors.
- R_OVERLAY_ECHO—Select plane to contain cursor.
- R_TRANSPARENCY_INDEX—Specify HP 98720 transparency index.
- R_UNLOCK_DEVICE—Unlock device.
- READ_COLOR_MAP—Read color map.
- SWITCH_SEMAPHORE—Semaphore control.
- TRANSPARENCY—Allow "screen door" for transparency pattern.
- ZWRITE_ENABLE—Allows creation of 3D cursors in overlay.

The following gescape functions are unique to this driver and are discussed in the following pages.

- ZBUFFER_ALLOC—Allocates frame buffer memory for Starbase.
- ZSTATE_RESTOR—Allows creation of 3D cursors in overlay.
- ZSTATE_SAVE—Allows creation of 3D cursors in overlay.

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 HP98721-5. 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

```
/* 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

```
real arg1(64),arg2(64)
arg1(1)=6
call gescape(fildes,ZBUFFER_ALLOC,arg1,arg2)
```

Pascal Syntax

```
{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(fi1des,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

```
/* 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

```
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

```
{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(fi1des,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

```
/* 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

```
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

```
{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);
```

Performance Tips

1. If only the HP 98721 driver and the ITE are accessing the graphics device it is safe to turn off the semaphore operations. This can result in a 10 to 20 percent speed increase. If a tracking process is initiated, semaphores will automatically be turned on. While in most cases the system will work with the tracking process running and semaphores turned off, there is a chance that continuous movement of the cursor could halt the graphics accelerator for a significant period of time. If this is not a problem, semaphores may be turned off after tracking is initiated.

2. As with any driver, buffering is done to enhance performance. If `buffer_mode` is turned off or if an inordinate amount of `make_picture_current` calls are done then performance can be degraded substantially.

3. Performance optimizations have been made so that sequential calls of the same output primitive with no intervening attribute changes or different primitive calls go faster. For example, the sequence `polygon, polygon, polyline, polyline` is faster the `polygon, polyline, polygon, polyline`. Also, `line_color, polyline, polyline` is faster than `line_color, polyline, line_color, polyline`. So grouping by primitive and subgrouping primitives by attribute can give substantial performance improvements.

4. Screen clears will be significantly faster if the area to be cleared starts on a 128-pixel boundary and is some multiple of 128 pixels wide. This can be checked by using the Starbase routines `transform_point` and `vdc_to_dc` to convert the bounds of the clear rectangle to device coordinates. Screen clears to the default `vdc_extent` will be aligned. Screen clears are also much faster when the background color index is zero. Screen clears with a non-zero index require two passes, resulting in slower performance.

5. When doing shaded polygons, the fewer the features, the faster the polygon generation. Positional viewpoint and light sources can significantly degrade performance.

6. If Starbase echoes are in the overlay plane then graphics performance is significantly better since it is not necessary to "pick up" the cursor each time the frame buffer is updated.

7. The procedure `block_move` is faster than `block_read` or `block_write` since the hardware frame buffer block mover can be used.

8. The performance of `block_read` and `block_write` is significantly better if both the source and destination begin on the same byte boundary since data can be transferred 32 bits at a time rather than one byte at a time. For example, one way to ensure this condition is to define pixel arrays as type short (16-bit integers) and start `block_read` and `block_write` actions on even pixels only. This can more than double performance.

9. With dithering, shading, and Z-buffering off, the SRX rendering engine runs at full speed while rendering flat shaded polygons. These three rendering techniques slow the rendering of polygons on the SRX. This is especially noticeable on large polygons. Turning on any one of the three could noticeably lower the rendering performance.

   When using the full 24 planes, dithering is turned off by default, and 12/12 double buffering will turn dithering on by default. To turn dithering off again, use `fill_dither` (`fildes`, 1).

   Using the pattern `gescape` or replacement rules that require extra reads of the frame buffer (e.g. ⟨*source*⟩ *or* ⟨*destination*⟩) will also degrade performance. It takes time to do the extra reads.

10. Typically, the SRX rendering engine renders primitives from its internal buffer as the system CPU is doing other things. Substantial performance benefits can be realized from this parallel processing.

    However, certain operations will cause the CPU to wait for the SRX to finish emptying its buffer. An example of this wait is the `make_picture_current` operation. Also, any operation that reads information from the SRX will cause this wait to occur. Following are some typical operations that read values from the SRX:

    a. Many two-dimensional primitives used in three-dimensional mode read the Z value from the SRX. The following primitives are examples: `text2d`, `polymarker2d`, `arc`, `ellipse`, and `spline_curve2d`. The solution is to always use three-dimensional primitives when in three-dimensional mode.

    b. Two operations read the matrix values from the SRX: `pop_matrix2d` and `pop_matrix3d`. If the values in the popped

matrix are not needed, use `pop_matrix`, which does not cause any information to be read from the SRX.

Cautions

The following cautions are provided in using this driver:

1. As mentioned previously, accessing the off-screen portion of the frame buffer (using the `gescape` functions) should be done with care since other processes access this region. The HP 98720 and HP 98721 drivers use a 128×1024 strip of off-screen memory that begins at (1920, 0). The HP 98721 driver in particular uses the rectangular area of 64×196 located at (1984, 828). This area is used to store the fill pattern when in `CMAP_NORMAL` mode and three 64×64 areas for storing the raster cursor, raster cursor transparency pattern, and the saved raster. If the HP 98721 driver is not being used in `CMAP_NORMAL` mode, raster cursors are not being used in the graphics planes and no HP 98720 drivers are opened to the graphics planes, the area can be safely used for more zbuffer or other purposes. If the HP 98721 driver is opened to the overlay planes, it is not recommended that any of the overlay off-screen be used. The overlay off-screen contains the ITE font (which is regenerated when control-shift-reset is done on the ITE keyboard) and may contain any number of window system fonts depending on the current window usage.
2. Polygons of up to 255 vertices are supported. If a polygon has more than 255 vertices, only the first 255 vertices are displayed.
3. Certain `gescape` functions should be used with caution since they bypass protection mechanisms used to prevent multiple processes from interfering with each other. For example, since the hardware resources can only be rationally used by one graphics process at a time, the driver activates a semaphore and locks the device before doing any output. This ensures, for example, that process A will not change the replacement rule while process B is in the middle of filling a polygon. It also prevents the terminal (`tty`) driver from overwriting any graphics processes that are outputting to the device. The driver unlocks the device when finished processing output. Some of the `gescape` functions listed in this chapter allow the

user to change this locking mechanism and should be used with *great caution.*

4. When using the HP 98720 device with a graphics accelerator it is possible for illegal operations to cause the transform engine or scan converter hardware to enter an unknown state. If this happens, Starbase will report an error the next time it tries to use the hardware. The user will see this as a `Transform engine timed out` or `Hardware/scan_converter time out` error. These are Starbase errors 14 and 52 respectively. This is a very serious error condition. If the HP 98721 Device Driver is being used, this is a fatal error. When this error is discovered, Starbase reports the error and aborts execution.

   If an application needs to take some emergency action before an untimely termination, such as saving valuable data, the application should check for these error conditions and take appropriate measures. Errors may be caught by an application using the `gerr_control` procedure described in the *Starbase Reference* manual.

   It is also possible to avoid the termination completely if the application's error handler does not return control to Starbase. It is, however, impossible to proceed with any graphics efforts using the accelerator.

   If the HP 98720 or HP 98720w drivers are being used to access the hardware and if they detect such an error, they will report the error condition, reset the transform engine, and continue (since they do not use the accelerator hardware).

Contents

HP98730

The HP 98730 Device Driver

Device Description

This graphics display station includes an HP 98730A Graphics Controller, a high resolution 16 or 19 inch color display (purchased separately), an optional accelerator and Z-buffer, and optionally 8, 16, or 24 planes of frame buffer memory. The graphics controller plugs into an I/O slot of the SPUs. (See the "Introduction" section of this manual for systems supporting this controller.)

Two device drivers are provided to access the HP 98730 display:

- HP 98730—The HP 98730 Device Driver is used to access the graphics display without using the optional graphics accelerator. Access can be with or without HP Windows 9000 and the X Window System.
- HP 98731—The HP 98731 Device Driver is used to access the graphics display using only the optional graphics accelerator, with or without HP Windows/9000 and the X Window System.

This section covers the HP 98730 Device Driver, see the "HP 98731 Device Driver" section for information on using the HP 98731 driver.

The display has a resolution of 1280×1024 pixels. The standard color display system has eight planes of frame buffer to provide 256 simultaneous colors. You can add optional memory in banks of eight planes each. A fully configured system consists of three banks of frame buffer for full 24 bit per pixel color, dedicated boards for full 16 bit Z-buffer capability with graphics acceleration, and four overlay planes for non-destructive alpha, cursors, or graphics.

An 8-plane configuration allows 256 colors to be displayed simultaneously from a pallet of 16 million. A 16-plane system is like two 8-plane frame buffers where only one 8-plane buffer is displayed at any time. This configuration is useful for double buffering. When three banks of frame buffer are installed, the system may be configured to display eight bits red, eight bits green and eight bits blue per

pixel. Double buffering may also be achieved at a resolution of four bits red, four bits green and four bits blue.

The display system is a bit-mapped device with special hardware for:

- Write enable/disable individual planes.
- Video enable/disable individual planes.
- Memory writes with specified replacement rule (see `drawing_mode` in *Starbase Reference* manual).
- Video blinking of individual planes.
- Video blinking of individual color map locations.
- Arbitrary sized rectangular memory to memory copies.
- Pixel pan and zoom.
- Analog blending of frame buffer outputs.
- Raster and vector cursors.

The display is organized as an array of bytes, with each byte representing a pixel on the display. (On Series 800 systems the display can be accessed on a 32-bit word/pixel basis.) When eight planes are installed, color map indexes range from 0–255. The color map is a RAM table that has 16 or 256 addressable locations and is 24 bits wide (eight bits each for red, green and blue). Thus, the pixel value in the frame buffer addresses the color map, generating the color programmed at that location. (32-bit word/pixel basis.) When eight planes are installed, color map indexes range from 0–255. The color map is a RAM table that has 16 or 256 addressable locations and is 24 bits wide (eight bits each for red, green and blue). Thus, the pixel value in the frame buffer addresses the color map, generating the color programmed at that location.

In addition to the frame buffer banks of eight planes each, 4-overlay planes are provided. These overlay planes have their own unique color map, separate from the color map used for the image planes. This color map consists of sixteen 24-bit entries, allowing the user to select sixteen colors from the full pallette of over 16 million choices. In addition, each entry in the overlay color map may be set to be `dominant`, `non-dominant`, or `blended` with the image planes.

A `dominant` entry causes all pixels in the overlays set to that value to display the color in the overlay map, regardless of values in the image planes "below"

it. A `non-dominant` entry causes pixels with that value to display the color in the image planes "below". A `blended` entry will cause the analog color output from the overlays to be summed with the analog output from the image planes. Color values are clamped to their full value of 1.0 if the sum would exceed this saturation value.

You can use overlay planes for non-destructive alpha, graphics, or cursors. For example, on displays that run it, the Internal Terminal Emulator (ITE) uses three of the overlay planes for alpha information. This way there is no interaction between ITE text and images in the graphics planes. Windows/9000 also runs in the overlay planes. The X Window system uses both the overlay and image planes. To do graphics in the overlay planes the HP 98730 Device Driver may be opened directly to the overlay planes as if they were a separate device (refer to "Setting up the Device" in this driver for more information).

Typically, the user does not need to directly read or write pixels in the frame buffer. However, for those applications which require direct access, Starbase does provide the `gescape` function `R_GET_FRAME_BUFFER`, which returns the virtual memory address of the beginning of the frame buffer (this `gescape` is discussed in the appendix of this manual). Frame buffer locations are then addressed relative to the returned address. The first byte of the frame buffer (byte 0) represents the upper left corner pixel of the screen. Byte 1 is immediately to its right. Byte 1279 is the last (right-most) pixel on the top line. The next 768 bytes of the frame buffer are not displayable. Byte 2048 is the first (left-most) pixel on the second line from the top. The last (lower right corner) pixel on the screen is byte number 2,096,383.

If more than one bank of optional frame buffer is installed then bank switching must be used to access the additional memory. A number of Starbase calls may set the bank register so it is advisable to call `bank_switch` just prior to making accesses to the frame buffer pointer to ensure desired results.

The off-screen portion of the frame buffer may be accessed via the `gescape` function `R_FULL_FRAME_BUFFER` also documented in the appendix. Care should be taken when using this `gescape` since other processes, Starbase, and the window system access the frame buffer off-screen memory.

Setting Up the Device On Series 300

The HP 98730 Device Driver can be used with the graphics display configured in either internal or external DIO-I address space, or in DIO-II address space. Refer to the *Configuration Reference Manual* for a description of internal and external DIO-I address space and DIO-II address space.

Note If the HP 98730 is configured as an external display, there will *not* be an Internal Terminal Emulator (ITE) for that device. Since it is the ITE that normally initializes the display, external devices must be reset after power-up by running a simple Starbase program with a mode of `RESET_DEVICE` in the `gopen` call. It may also be necessary to run this program after running an application which manipulated the overlay color map, such as a windows application program. An example program which could be called from `/etc/rc` during power-up is given at the end of this section. For more details concerning the effects of `RESET_DEVICE`, see the "Device Initialization" information in this section.

The Graphics Interface card may be installed in any DIO slot in the computer's backplane or in any I/O slot of the expander.

DIO-I Switch Settings

The graphics interface card has a single 8-bit address select switch. Looking at the switches so that the dot is in the lower left corner, the leftmost switch is labeled DIO1 to the bottom (0), and DIO2 to the top (1). To configure the system in DIO-I space, this switch must be set to the DIO1 (0) position. The next switch to the right is labeled INT and determines if the HP 98730 workstation is configured as an internal or external display. In addition, the next six switches to the right are labeled for select code determination (five of the six switches are actually used for the select code). There is also a jumper labeled JP1.

The frame buffer uses two megabytes of I/O address space, starting at FB_BASE. The jumper (JP1) determines the address of FB_BASE.

JP1 is set to $200K FB_BASE address is $200 000

JP1 is set to $800K FB_BASE address is $800 000

Systems which use the HP 98730 display as a DIO-I system console will map the frame buffer to $200 000; systems which use the display as an external DIO-I device will map the frame buffer to $800 000.

The control space requires 128 Kbytes of space, starting at CTL_BASE. The six switches labeled SC determines the address of CTL_BASE. The HP 98730 may be configured as an external display or as an internal display. Since only 64 Kbytes of space is normally allotted for external I/O select codes, two consecutive select codes will be used when configuring the device as an external display.

The following table lists the binary switch settings with the corresponding values of CTL_BASE for external I/O settings. The table also lists the select codes that are used for each setting.

Table HP98730-1. DIO-I Control Space Settings (External I/O)

Switch Setting MSB to LSB	CTL_BASE	DIO-I Select Code
01101010	$6A0000	10–11
01101100	$6C0000	12–13
01101110	$6E0000	14–15
01110000	$700000	16–17
01110010	$720000	18–19
01110100	$740000	20–21
01110110	$760000	22–23
01111000	$780000	24–25
01111010	$7A0000	26–27
01111100	$7C0000	28–29
01111110	$7E0000	30–31

For a system console (internal) the switch setting is 01010110 and the CTL_BASE is $560 000.

If the HP 98730 is configured as the system console, the CTL_BASE needs to be placed at $560 000 and the JP1 must be open (no jumper—or jumper is on one pin), which is an interal I/O setting. If the device is not used as the system console, then the control space should not be placed in internal I/O space. It is likely to overlap the address space of other system hardware. In this case, an external I/O space setting should be selected with two consecutive select codes which are not used by the system.

DIO-II Switch Settings

If the left-most switch is set to DIO2 (1), the HP 98730 device can be used in DIO-II address space. In this mode, the next seven switches determine the DIO-II select codes to be used. An HP 98730 device will use three DIO-II select codes. Both the frame buffer and control space reside in the select code areas, so the jumper JP1 is ignored.

The control space requires 4 Mbytes of space, starting at CTL_BASE. The seven switches labeled "SC" at the top of the select switch determine the address of CTL_BASE. The frame buffer requires 8 Mbytes of space, starting at FB_BASE.

Table HP98730-2. DIO-II Control Space Settings

Switch Setting MSB to LSB	CTL_BASE	DIO-II Select Code	FB_BASE
10000101	$01400000	133	$01800000
10001001	$02400000	137	$02800000
10001101	$03400000	141	$03800000
10010001	$04400000	145	$04800000
10010101	$05400000	149	$05800000
10011001	$06400000	153	$06800000
10011101	$07400000	157	$07800000
10100001	$08400000	161	$08800000
10100101	$09400000	165	$09800000
10101001	$0A400000	169	$0A800000
10101101	$0B400000	173	$0B800000
10110001	$0C400000	177	$0C800000
10110101	$0D400000	181	$0D800000
10111001	$0E400000	185	$0E800000
10111101	$0F400000	189	$0F800000
11000001	$10400000	193	$10800000
11000101	$11400000	197	$11800000
11001001	$12400000	201	$12800000
11001101	$13400000	205	$13800000
11010001	$14400000	209	$14800000
11010101	$15400000	213	$15800000
11011001	$16400000	217	$16800000
11011101	$17400000	221	$17800000
11100001	$18400000	225	$18800000
11100101	$19400000	229	$19800000
11101001	$1A400000	233	$1A800000
11101101	$1B400000	237	$1B800000
11110001	$1C400000	241	$1C800000
11110101	$1D400000	245	$1D800000
11111001	$1E400000	249	$1E800000
11111101	$1F400000	253	$1F800000

DIO-II displays may be used as the system console or as external displays. In order to use the display as system console, it must be configured as the first

DIO-II display in the system, and there must be no DIO-I console, or remote terminals. Being the first DIO-II device means that it has the lowest DIO-II select code in the system. In order to use a HP 98730 device as a DIO-II system console, select code 133 is recommended.

Note

It is necessary to increase some of the HP-UX tunable system parameters due to the size of the DIO-II mapping of an HP 98730 device. For details on how to reconfigure your kernel, refer to the *HP-UX System Administrator Manual* (particularly the "Configuring HP-UX" section in "The System Administrators Toolbox" and the "System Parameters" appendixes.

It is *essential* that you consult the above referenced HP-UX documentation before you attempt to reconfigure your system. It is possible to adversely affect your HP-UX system if a mistake is made. Ensure you have an understanding of these procedures before proceding.

In order to run an HP 98730 device in DIO-II the following system parameters must have these minimum values:

```
shmmax        0xC00000

shmmaxaddr    0x1800000

   dmmin      16

   dmmax      1024

   dmtext     1024

   dmshm      1024
```

Example Program to Reset the HP 98730

```
/*
 * Starbase program: reset98730.c
 * Compile: cc -o reset98730 reset98730.c -ldd98730 -lsb1 -lsb2
 * Destination: /usr/bin
 * Execute: add line to the /etc/rc - "/usr/bin/reset98730 /dev/crt.external"
 *
 * Example program to be put in /etc/rc for resetting an external HP 98730
 *  device during power-up.
 */
#include <starbase.c.h>

main(argc,argv)
int argc; char *argv[];
{
   int fildes;

   if ((fildes = gopen(argv[1],OUTDEV,"hp98730",INIT|RESET_DEVICE)) < 0)
      printf("External HP 98730 %s initialization failed.\n",argv[1]);
   else {
      printf("External HP 98730 %s initialization succeeded.\n",argv[1]);
      gclose(fildes);
   }
}
```

Setting Up the Device on the Series 800

Up to four HP 98730 devices can be connected to a Series 800 SPU using four A1017A interface cards. However, it is recommended that only two HP 98730 devices have the Internal Terminal Emulator (ITE) or window systems running on them. With the A1047A interface, only two devices can be connected, both of which may run an ITE.

The Series 800 ITE supports power-fail recovery on the HP 98730 device, but Starbase does not support this feature. If you want to support power fail, you must catch the power-fail signal and save any Starbase state needed. Then, `gclose` the device and `gopen` the device again when the power turns on.

Address Space Usage On Series 300

The HP 98730 device is memory mapped into a processes virtual address space, starting at the value specified by the environment variable `SB_DISPLAY_ADDR`. If this variable is not set, then mapping defaults to `0xB00000`. The control space starts at this address and grows towards larger address values. After the control space comes the frame buffer, then shared memory mapped for Starbase drivers. The size of the address space used for control space and the frame buffer depends on whether the device is used in DIO-I or DIO-II. In DIO-I, control space consumes 128 Kbytes and the frame buffer uses 2 Mbytes. In DIO-II, control space is 4 Mbytes and the frame buffer is 8 Mbytes. The size of the Starbase drivers' shared memory is always the same, and is slightly less than 300 Kbytes.

If your application maps memory pages to specific addresses, or needs a large stack, then you may need to adjust `SB_DISPLAY_ADDR` to avoid conflicts.

Special Device Files (mknod) On Series 300

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 `crt`.

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

When the device is at the internal DIO-I address (refer to the "Switch Settings" section) the `mknod` parameters should create a character special device with a

major number of 12 and a minor number of 0. Note that the leading 0x causes the number to be interpreted hexadecimally.

```
mknod /dev/crt c 12 0x000000
```

When the device is at an external DIO-I or any DIO-II address (refer to the "Switch Settings" section) the mknod parameters should create a character special device with a major number of 12 and a minor number of 0x⟨*sc*⟩0200 where ⟨*sc*⟩ is the two-digit external select code in hexadecimal notation.

```
mknod /dev/crt c 12 0x⟨sc⟩0200
```

The HP 98730 Device Driver may also be used for the overlay planes in graphics mode. The minor number may be set to cause Starbase drivers to use either three or four overlay planes. When running to three planes, one plane is still reserved for cursors. When running to all four overlays, only the hardware cursor is available for Starbase graphics echoes. If more than one echo is requested, or if another process is using the cursor, the request for another echo will fail. Note that since the terminal emulator and window system operate in the overlay planes also, there will be interactions with these processes if a graphics driver is opened in this manner while these processes are present. To open the HP 98730 Device Driver to three overlay planes instead of the graphics planes, the last byte of the minor number must be one. To run to all four overlays, the last byte of the minor number must be three.

For example, when the device is at an internal DIO-I address, the mknod parameters for the overlay device, with one plane reserved for cursors, should create a character special device with a major number of 12 and a minor number of 1.

```
mknod /dev/ocrt c 12 0x000001
```

To create a device file for all four overlays, the command would be:

```
mknod /dev/o4crt c 0x000003
```

When the device is at an external DIO-I address or any DIO-II address (refer to the section on "Switch Settings") the `mknod` parameters for the same device should create a character special device with a major number of 12 and a minor number of 0x⟨*sc*⟩0201 or 0x⟨*sc*⟩0203 where ⟨*sc*⟩ is the two-digit select code.

```
mknod /dev/ocrt  c  12  0x⟨sc⟩0201
```

or

```
mknod/dev/ocrt c 12 0x⟨sc⟩0203
```

Special Device Files (mknod) On the Series 800

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 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 names that are suggested for the devices are `crt`, `crt0`, `crt1`, or `crt2`.

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

When creating the device file the `mknod` parameters should create a character special device with a major number of 14 and a minor number of the format below (where ⟨*lu*⟩ is the two-digit hardware logical unit number):

```
mknod /dev/crtx c 14 0x00⟨lu⟩00
```

The HP 98730 Device Driver may also be opened to the overlay planes in graphics mode. If the last byte of the minor number is one, 3-overlay planes are used for graphics (and the fourth plane is reserved for cursors for processes running in the image planes). If the last byte of the minor number is three, 4-overlay planes are used for graphics. Since the ITE and window system operate in the overlay

planes also, there will be interactions with these processes if a graphics driver is open in this manner while these processes are present.

To open all 4-overlay planes when the device is at the internal address, the `mknod` parameters should create a character special device with a major number of 14 and a minor number of three.

```
mknod /dev/ocrt4 c 14 0x00⟨lu⟩03
```

To open three overlay planes when the device is at the internal address, the `mknod` parameters should create a character special device with a major number of 14 and a minor number of one.

For example, the `mknod` parameters for a 3-plane overlay device should create a character special device with a major number of the format indicated below (where ⟨*lu*⟩ is the hardware logical unit number):

```
mknod /dev/ocrtx c 14 0x00⟨lu⟩01
```

Linking the Driver

The HP 98730 Device Driver is located in the `/usr/lib` directory with the file name `libdd98730.a`. This device driver may be linked to a program using the absolute path name `/usr/lib/libdd98730.a` or an appropriate relative path name, or by using the `-l` option `-ldd98730`. For example: to compile and link a program for use with this driver, use:

```
cc example.c -ldd98730 -lsb1 -lsb2 -oexample
fc example.f -ldd98730 -lsb1 -lsb2 -oexample
pc example.p -ldd98730 -lsb1 -lsb2 -oexample
```

Usage and Restrictions

When a device file for the overlay planes is used at gopen time. Bank switching is not supported.

Windows/9000 and graphics applications that want to talk to a graphics window may use this device driver. If the graphics window is in the overlay planes, this device driver does not support:

- Bank switching.
- Z-buffering.
- double buffering when using three overlay planes. (Double buffering in a window in the overlay planes is supported to 4-overlay planes. Refer to HP *Windows/9000 Documentation* for double buffering in windows.)
- Shading.
- The transform engine.

If the graphics window is in the image planes, this device driver does not support:

- Z-buffering.
- Shading.
- The transform engine.

Refer to the HP *Windows/9000 Programmer's Manual* for information on graphics windows in the image plane.

Graphics applications that want to talk to a local X window can use this device driver. If the window is in the overlay planes, this device driver does not support:

- Bank switching.
- Z-buffering.
- Double buffering.
- Shading.
- The transform engine.

If the graphics window is in the image planes, this device driver does not support:

- Z-buffering.
- Shading.
- The transform engine.

Transparency Index

There are four overlay planes in the HP 98730 display. Even though these planes can display 16 colors simultaneously, only 15 are available because one color is reserved for the transparency color. By default, this color is index 7 or 15, depending on the overlay depth. When the transparency color's index is written into the overlay planes, the observed color is that of the image planes. The transparency color is set when the X11 server is started and cannot be changed until the server is shut down.

HP Windows/9000 See-Thru Color

Windows/9000 runs in the overlay planes on the HP 98730 Display Station and provides the following:

- Applications can run in the image planes independent of Windows/9000.
- Applications can create windows in the image planes via Windows/9000 and also have graphics windows in the overlay planes. (Graphics windows created in the image planes still have their borders in the overlay planes). Refer to the HP *Windows/9000 Programmer's Manual* for information on creating windows in the image planes.
- Windows/9000 can provide both opaque and transparent backgrounds.

The see-thru facility allows you to create a transparent window. Refer to the see-thru information in Windows/9000 documentation.

By default index 3 (yellow) is reserved as see-thru. The HP 98730 Device Driver recognizes the Starbase environment variable `SB_OV_SEE_THRU_INDEX` that will allow the user to set the see-thru color map index to some other value. This environment variable will only have effect when using Windows/9000 or when running the program on the `raw` device. Programs running in graphics windows in the overlay planes use the `SB_OV_SEE_THRU_INDEX` value in effect at Windows/9000 startup time. Resetting this variable has no effect on already active programs using this driver. Any value out of the range 0–7, (or 0–15) if the window system was started to four overlay planes instead of three) will be ignored except for -1. The value -1 can be used to force no see-thru color map entries to be defined. If this environment variable is not set, or is set to an illegal value, the driver will default to using index 3 as see-thru.

When running the `raw` device, an explicit call to `define_color_table` will cause see-thru entries to be set back to `dominant`. When running to an overlay graphics window, an explicit call to `define_color_table` will preserve the see-thru entry currently defined to the window system.

Since the Term0 Server, Graphics Server, and Window Manager all use the HP 98730 driver to talk to this display, this environment variable also affects them. For example, the following situation will cause your Term0 text to seem to disappear:

1. A program sends color escape sequences to Term0 to get yellow-on-black characters.
2. The see-thru index corresponds to what used to be yellow (3).
3. The image planes are cleared to black so that see-thru shows black.

In another example, suppose your Windows/9000 environment variables are set up to have your window borders cyan-on-black. The borders will appear invisible if:

1. You set `SB_OV_SEE_THRU_INDEX=5` (cyan) before powering up Windows/9000.
2. The image planes are cleared to black so that see-thru shows black.

X Window System See_Thru Color

The X Window system always uses color 7 (15 for 4-plane devices) as the see-thru color. This cannot be changed.

Cursors

If no processes have opened all four overlay planes, then the fourth overlay plane is used for overlayed software cursors either by the HP 98730 or the HP 98731 drivers running in the image planes. The HP 98730 driver running in the overlay planes never uses the fourth overlay plane for cursors. Instead, either the hardware cursor or all three (four) overlay planes are used for cursors.

You can think of the fourth overlay plane used for cursors as a separate "cursor plane". Any data in the cursor plane will be displayed *over* data in the graphics planes. Data in the other three overlay planes will be displayed *over* data in the graphics planes and the cursor plane. For example, suppose a graphics application

is running in the graphics planes while the window manager is running in three of the overlay planes. If the application has a Starbase cursor in the overlay cursor plane, the cursor will always be visible inside regions of see-thru because the cursor has display priority over the graphics. If the cursor is moved outside of regions of see-thru, it is not visible since the non-see-thru regions in the overlay planes have display priority over the cursor plane.

The HP 98730 Display Station also supports a hardware cursor that supports all Starbase echo types. The hardware cursor is drawn to a fifth and sixth overlay plane accessible only by the hardware cursor. There is only one hardware cursor available. Usage of the hardware cursor is defined as follows:

1. If an application is running in a Starbase environment only (that is, Windows/9000 or the X Window system is not running), the hardware cursor is given to the first process that attempts to use cursors.
2. The X Window system sprite always uses the hardware cursor.
3. By default, if Windows/9000 is active then the window system sprite gets usage of the hardware cursor. The user may prevent the window system from using the hardware cursor by setting the `WMCONFIG` environment variable appropriately. See the HP Windows/9000 documentation for details.
4. Via the `gescape R_ECHO_CONTROL`, there is a mechanism for the user to control usage of the hardware cursor. This `gescape` is discussed in the appendix.

If the hardware cursor is already being used by another process, then software cursors are used by the HP 98730 driver. The user can control if the software cursors are overlayed in the fourth overlay plane or reside in the same planes currently being used for graphics by the `gescape R_OVERLAY_ECHO`. Refer to the appendix for a discussion of this `gescape`.

If a users application never uses cursors, the driver will never attempt to allocate the hardware cursor. However, once the driver has allocated the hardware cursor, the driver does not relinquish usage of the hardware cursor until `gclose` time.

If allocation of the hardware cursor was not successful, resources for the software cursor area are allocated (that is, offscreen areas for raster echo definitions). Once resources for software cursors have been allocated, the driver always uses software cursors and never again attempts to use the hardware cursor.

The following functions will cause the driver to attempt to allocate cursor resources (that is, either the hardware cursor or software cursor resources):

- `echo_type` or `define_raster_echo`.
- any of the `gescapes` `R_DEF_ECHO_TRANS`, `R_ECHO_MASK`, `R_ECHO_FG_BG_COLORS`, and `R_OV_ECHO_COLORS`.

Device 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/crt`.

Kind — Indicates the I/O characteristics of the device. This parameter may be one of the following:

- `INDEV`, Input only.
- `OUTDEV`, Output only.
- `OUTINDEV`, Input and Output.

Input may be done with this driver only when opened to an HP Windows/9000 or X Window system window.

Driver — The character representation of the driver type. This must be hp98730. For example:

`"hp98730"`	*for C.*
`'hp98730'//char(0)`	*for Fortran77.*
`'hp98730'`	*for Pascal.*

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.

- SPOOLED, cannot spool raster devices.
- MODEL_XFORM—Shading is not supported for this device. However, opening in MODEL_XFORM mode will affect how matrix stack and transformation routines are performed.
- 0—Open the device without clearing the screen. This will set the color map mode to CMAP_NORMAL, but will not initialize the color map itself. This will also disable **blending** if it was left enabled. This mode will not affect pixel panning and zooming.

  In an X Window the color map mode is initialized consistent with the X color map.
- INIT—open and initialize the device as follows:
  1. Clear frame buffer to 0s.
  2. Reset the color map to its default values.
  3. Enable the display for reading and writing.
  4. Restore pixel pan and zoom hardware for normal viewing, if opened to the image planes.
  5. In an X Window a new color map is created.
- RESET_DEVICE—open and reset the device as follows:
  1. Clear frame buffer and overlays to 0s.
  2. Reset the color map to its default values.
  3. Clear the overlay color map.
  4. Enable the display for reading and writing.
  5. Restore pixel pan and zoom hardware for normal viewing, if opened to the image planes.

6. Reset the graphics accelerator.

Note that the RESET_DEVICE flag bit should be used with caution: it will adversely affect any other processes using the device. This flag bit is intended to reset a device completely: this should only be necessary for devices in an unknown state, such as a device powered up in an external I/O space. Most programs should not use this flag bit.

Syntax Examples

To open and initialize an HP 98730 device for output:

For C Programs:

```
fildes = gopen("/dev/crt",OUTDEV,"hp98730",INIT);
```

For FORTRAN77 Programs:

```
fildes = gopen('/dev/crt'//char(0),OUTDEV,'hp98730'//char(0),INIT)
```

For Pascal Programs:

```
fildes = gopen('/dev/crt',OUTDEV,'hp98730',INIT);
```

Special Device Characteristics

For Device Coordinate operations, location $(0, 0)$ is the upper-left corner of the screen with X-axis values increasing to the right and Y-axis values increasing down. The lower-right corner of the display is therefore $(1279, 1023)$.

Offscreen Memory Usage

Offscreen memory is managed by a global resource manager to insure that multiple processes do not step on each other when using the offscreen. Offscreen is used by the device driver for:

- polygon fill patterns
- raster fonts
- raster echo definitions (if software cursors are used)

The offscreen memory is not allocated for any of the above functions unless the function is used. Therefore, if an application never does filled polygons, never

uses software cursors, and never uses raster fonts; the driver does not use the offscreen memory. Refer to the `gescape R_OFFSCREEN_ALLOC` for information on using the offscreen areas for personal use.

Device Defaults

Number of Color Planes

When the `gopen` procedure is called, this driver asks the device for the number of color planes available. This number can be either 3 or 4 (if running to the overlay planes), 8, 16, or 24. The device driver then acts accordingly.

Dither Default

The default number of colors searched for in a dither cell is 2. The number of colors allowed in a dither cell is 1, 2, 4, 8 or 16. For devices having 24 or more planes in `CMAP_FULL` mode (see `shade_mode`) dithering is not supported since full 24-bit color is available. If you are double buffering with 12 planes per buffer then the number of colors allowed in a dither cell is 1, 2, or 4.

Raster Echo Default

The default raster echo is the 8×8 array:

255	255	255	255	0	0	0	0
255	255	0	0	0	0	0	0
255	0	255	0	0	0	0	0
255	0	0	255	0	0	0	0
0	0	0	0	255	0	0	0
0	0	0	0	0	255	0	0
0	0	0	0	0	0	255	0
0	0	0	0	0	0	0	255

The maximum size allowed for a raster echo is 64×64 pixels. The default drawing mode for the raster echo is 7 (*or*).

If the driver does not have access to the hardware cursor, by default the raster echo is written to the same planes currently being used for graphics. For example, if the HP 98730 driver was opened to the image planes, the image planes are used for raster cursors. If the HP 98730 driver was opened to three overlay planes, the those three overlay planes are used for raster cursors. The location of software

raster and software non-raster cursors can be changed using the gescape function R_OVERLAY_ECHO.

Color Planes Defaults

In a raw display or HP Windows/9000 window, the default configuration is an 8-plane color mapped system regardless of the number of frame buffer banks installed. In an X Window the color plane definition is consistent with the X color map.

All planes in first bank are display enabled. All planes in first bank are write enabled.

Semaphore Default

Semaphore operations are enabled.

Line Type Defaults

The default line types are created with the bit patterns shown below:

Table HP98730-3.

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

Default Color Map

If the fourth gopen parameter is zero (0), the current hardware color map is used on raw or HP Windows/9000 color displays. On X Windows the windows current color map is used.

If the fourth gopen parameter is INIT, the current color map is initialized to the default values shown in the following table.

Table HP98730-4. 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

Use the inquire_color_table procedure to see the rest of the 255 colors.

When INIT is used in the shade_mode procedure call the color map will be initialized dependent on the mode parameter and the number of frame buffer banks installed.

mode=CMAP_NORMAL — Only one bank of the three banks can be displayed at a time, unless video blending is enabled.

mode=CMAP_MONOTONIC The color map will be initialized as:

```
for (i=0; i<256; i++) {
   cmap[i].red = cmap[i].green = cmap[i].blue = i/255.0;
}
```

Only one bank of the three banks can be displayed at a time, unless video blending is enabled.

mode=CMAP_FULL With less than three banks installed the color map will be initialized as three bits red, three bits green and two bits blue. The three most significant bits are red and the two least significant bits are blue. Only one bank of the three banks can be displayed at a time.

With three or more banks installed the color map will be initialized as the CMAP_MONOTONIC case above, the first bank of eight will go through the blue portion of the color map, the second bank goes through the green portion and the third bank goes through the red portion. In this mode the color map is transparent and the eight bits from each bank drives the appropriate DAC. The color map could be subsequently modified in this mode to do things like gamma correction or double buffering of four bits per color.

Red, Green and Blue

Each file descriptor opened as an output device has a color table associated with it. If multiple file descriptors are open to the same device, the color table and the device's color map may not always be identical. The color table does not track the color map if the device's color map is changed by another file descriptor path.

For Starbase procedures that have parameters for red, green and blue, the way the actual color is chosen depends on the current **shade_mode** setting.

mode=CMAP_NORMAL The color map is searched for the color that is closest in RGB space to the one requested, and that color map index is written to the frame buffer for subsequent output primitives. It is more efficient to

select a color with an index rather than specifying a color with red, blue and green values in this mode due to the time it takes to figure out which index in the color table most closely matches the specified color.

`mode=CMAP_MONOTONIC` The red, green and blue value is converted to an intensity value using the equation:

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

This intensity is converted to an index value by mapping intensity 0.0 to the minimum index set by **shade_range** and intensity 1.0 to the maximum index set by **shade_range**. This mode is useful for displaying a high quality monochrome picture on an 8-plane system from data that produces a high quality color picture on a 24-plane system.

`mode=CMAP_FULL` The color values will be mapped directly to an index with the assumption the color map is setup to a predefined full color state.

Note Multiple gopen parameters of an X Window will share a single color map definition. See the *Starbase Programming with X11* manual for more information.

Starbase Functionality

Commands Not Supported

The following commands are ignored.

backface_control	depth_cue_range
bf_control	hidden_surface
bf_fill_color	light_ambient
bf_interior_style	light_attenuation
bf_perimeter_color	light_model
bf_perimeter_repeat_length	light_source
bf_perimeter_type	light_switch
bf_surface_coefficients	shade_range
bf_surface_model	surface_coefficients
define_trimming_curve	surface_model
depth_cue	viewpoint
depth_cue_color	zbuffer_switch

Commands Conditionally Supported

The following commands are supported under the listed conditions:

block_read, block write	The raw parameter for the block_read and block_write commands is normally ignored by this driver. To use the raw mode, you must call the R_BIT_MODE gescape discussed in the appendix of this manual.
pattern_define	4×4 is the largest supported pattern.
shade_mode	The color map mode may be selected but shading can not be turned on.
text_precision	Only STROKE_TEXT precision is supported.
vertex_format	The use parameter must be zero, any extra coordinates supplied will be ignored.

Fast Alpha and Font Manager Functionality

The HP 98730 Device Driver supports raster text calls from the fast alpha and font manager libraries. These calls may be made while running in the overlay or image planes. Since raster fonts consist of one byte per pixel, image plane raster text is written only to the currently selected bank. This is similar to the operation of other raster functions such as `block_write`. Fast alpha and font manager fonts can be optimized. See the *Fast Alpha/Font Manager's Programmer's Manual* for further information.

Parameters for gescape

The following `gescape` functions are common to two or more of the Hewlett-Packard displays supported by Starbase. Detailed information about these functions can be found in Appendix A.

- `BLINK_INDEX`—alternate between HP 98730 hardware color maps. This `gescape` is not supported while image blending is active. Refer to the `IMAGE_BLEND gescape`.
- `BLINK_PLANES`—blink the display (blink rate is 3.75 Hz for this device)
- `IMAGE_BLEND`—control analog blending of image plane frame buffer output
- `OVERLAY_BLEND`—control analog blending of overlay plane frame buffer output
- `PAN_AND_ZOOM`—do pixel panning and zooming
- `R_BIT_MASK`—bit mask
- `R_BIT_MODE`—bit mode
- `R_DEF_ECHO_TRANS`—define raster echo transparency
- `R_DEF_FILL_PAT`—define fill pattern
- `R_DMA_MODE`—changes definition of `raw` for block writes
- `R_ECHO_CONTROL`—control hardware cursor allocation
- `R_ECHO_FG_BG_COLORS`—define cursor color attributes

- R_ECHO_MASK—define a raster echo mask pattern
- R_FULL_FRAME_BUFFER—full frame buffer
- R_GET_FRAME_BUFFER—read frame buffer address
- R_GET_WINDOW_INFO—returns frame buffer address of window
- R_LINE_TYPE—define line style and repeat Length
- R_LOCK_DEVICE—lock 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 HP 98730 transparency index
- R_UNLOCK_DEVICE—unlock device
- READ_COLOR_MAP—read color map
- SET_BANK_CMAP—define bank color map to be used for image blending
- SWITCH_SEMAPHORE—semaphore control

PerformanceA Tips

1. If only one process is accessing the graphics display, it is safe to turn off the semaphore operations (see the SWITCH_SEMAPHORE gescape), and a 10 to 20 percent speed improvement can be obtained. If a tracking process is initiated, then semaphores will automatically be turned on.
2. As with any driver, buffering is done to enhance performance. Performance can be degraded if `buffer_mode` is turned off or an inordinate amount of `make_picture_current` calls are done.
3. Performance optimizations have been made so that sequential calls of the same output primitive, with no intervening attribute changes or

different primitive calls, go faster. For example the sequence `line_color`, `polyline`, `polyline` is faster than `line_color`, `polyline`, `line_color`, `polyline`. So grouping by primitive and subgrouping primitives by attribute can give some performance improvements.

4. If Starbase echos are overlayed (i.e. in the fourth overlay plane), or hardware cursors are used, graphics performance is significantly better since it is not necessary to "pick up" the cursor each time the frame buffer is updated.

5. Screen clears will be significantly faster if the area to be cleared starts on a 128-pixel boundary and is some multiple of 128-pixels wide. This can be checked by using the Starbase routines `transform_point` and `vdc_to_dc` to convert the bounds of the clear rectangle to device coordinates. Screen clears to the default `vdc_extent` will be aligned. Screen clears are also much faster when the background color index is zero. Screen clears with a non-zero index require two pases, which result in slower performance.

6. Polygons are filled faster when the drawing mode is ⟨*SOURCE*⟩, `NOT_SOURCE`, `ZERO`, or `ONE`.

7. Horizontal and vertical lines are faster than diagonal lines on this device since the hardware block mover is used to generate pixels.

8. The procedure `block_move` is faster than `block_read` or `block_write` since the hardware frame buffer block mover can be used.

9. Performance of `block_read` and `block_write` is significantly better if both the source and destination begin on the same byte boundary (since data can be transferred 32 bits at a time rather than one byte at a time). For example, one way to ensure this condition is to define pixel arrays as type short (16-bit integer), and start `block_read` and `block_write` on even pixels only. This can more than double performance.

10. `block_write` on Series 800 machines with the A1047A interface can go faster by using DMA. See `R_DMA_MODE gescape`.

Cautions

The following cautions are provided in using this driver:

1. As mentioned previously, accessing the off-screen portion of the frame buffer (using a `gescape` function) should be done with care, since other processes access this region. See the section on offscreen usage for details.
2. Certain `gescape` functions should be used with caution since they bypass protection mechanisms used to prevent multiple processes from interferring with each other. For example, since the hardware resources can only be rationally used by one graphics process at a time, the driver activates a semaphore and locks the device before doing any output. This ensures, for example, that process A will not change the replacement rule while process B is in the middle of filling a polygon. It also prevents the terminal (`tty`) driver from overwriting any graphics processes that are outputting to the device. The driver unlocks the device when done processing output. Some of the `gescapes` listed in this chapter allow the user to change this locking mechanism and should be used with *great caution.*

Contents

HP98731

The HP 98731 Device Driver

Device Description

The HP 98731A is an optional graphics accelerator and Z-buffer for the HP 98730A Display Controller. The graphics controller plugs into an I/O slot of the SPUs. (See the "Introduction" section of this manual for systems supporting this controller and accelerator.)

Two device drivers are provided to access the HP 98730 display:

- HP 98730—used to access the graphics display without using the optional graphics accelerator, with or without HP Windows/9000 or the X Window system.
- HP 98731—used to access the graphics display using only the optional graphics accelerator, with or without HP Windows/9000 or the X Window system.

This section covers the HP 98731 Device Driver; see the "HP 98730 Device Driver" section for information on using the HP 98730 driver.

The display has a resolution of 1280×1024 pixels. The standard color display system has eight planes of frame buffer to provide 256 simultaneous colors. You can add optional memory in banks of eight planes each. A fully configured system consists of three banks of frame buffer for full 24-bit per pixel color, a dedicated board for full Z-buffer capability, and 4-overlay planes for non-destructive alpha, cursors, or graphics. In order to use the HP 98731 Device Driver, the system must be configured with the graphics accelerator boards and at least one bank of eight planes.

An 8-plane configuration allows 256 colors to be displayed simultaneously from a pallet of 16 million. A 16-plane system is like two 8-plane frame buffers where only one 8-plane buffer is displayed at any time. This configuration is useful for double buffering. When three banks of frame buffer are installed, the system may

be configured to display eight bits red, eight bits green, and eight bits blue per pixel. Double buffering may also be achieved at a resolution of four bits red, four bits green, and four bits blue.

The display system is a bit-mapped device with special hardware for:

- Write enable/disable individual planes.
- Video enable/disable individual planes.
- Memory writes with specified replacement rule. (see `drawing_mode`)
- Video blinking of individual planes.
- Video blinking of individual color map locations.
- Arbitrary sized rectangular memory to memory copies.
- Write enable/disable of pixels in 4×4 cell for "screen door" transparency.
- Up to three VLSI NMOS III processors with hardware floating point for high speed three-dimensional transformations.
- NMOS III scan converter with six axis interpolation for Gouraud shaded, Z-buffered vectors and polygons.
- Pixel pan and zoom.
- Analog blending of frame buffer outputs.
- Raster and vector cursors.
- Pixel clipping for full speed graphics to obscured windows.
- Dedicated 2K by 1K 16-bit zbuffer.

The display is organized as an array of bytes, with each byte representing a pixel on the display. When eight planes are installed, color map indexes range from 0–255. The color map is a RAM table that has 256 addressable locations and is 24 bits wide (eight bits each for red, green, and blue). Thus, the pixel value in the frame buffer addresses the color map, generating the color programmed at that location.

In addition to the frame buffer banks of eight planes each, four overlay planes are provided. These overlay planes have their own unique color map, separate from the color maps used for the image planes. This overlay color map consists of sixteen 24-bit entries, allowing the user to select 16 colors from the full pallette

of over 16 million choices. In addition, each entry in the overlay color map may be set to `dominant`, `non-dominant`, or `blended` with the image planes.

A `dominant` entry causes all pixels in the overlays set to that value to display the color in the overlay map, regardless of values in the image planes "below".

Aa `non-dominant` entry causes pixels with that value to display the color in the image plane "below".

A `blended` entry will cause the analog color output from the overlays to be summed with the analog output from the image planes. Color values are clamped to their full value of 1.0 if the sum would exceed this saturation level.

By default, the HP 98731 Device Driver sets all overlay color map entries to be `dominant` when opened to the overlays. Entries may be set to be `non-dominant` with the Starbase `gescape R_TRANSPARENCY_INDEX`. Entries may be set to blend with the image planes by using the Starbase `gescape OVERLAY_BLEND`. See the descriptions of these `gescape` functions for more details.

You can use overlay planes for non-destructive alpha, graphics, or cursors. For example, when the HP 98730 is used as system console, the Internal Terminal Emulator (ITE) uses three of the overlay planes for alpha information. This way there is no interaction between ITE text and images in the graphics planes. Windows/9000 also runs in the overlay planes. The X Window system uses both the image and overlay planes. To do graphics in the overlay planes, HP 98731 Device Driver may be opened directly to the overlay planes, as if they were a separate device (refer to the section "Setting up the Device" for more information).

The HP 98730 display system provides one hardware cursor which supports all Starbase echo types. If more than one cursor is needed, one overlay plane can be used for graphic cursors. You can think of the overlay plane used for cursors as a separate "cursor plane." Any data in the cursor plane will be displayed over data in the graphics planes. Data in the other three overlay planes will be displayed over data in the graphics planes and the cursor plane.

For example, suppose a graphics application is running in the graphics planes while the window manager is running. If the application has a Starbase cursor in the overlay cursor plane, the cursor will always be visible inside regions of see-thru because the cursor has display priority over the graphics. If the cursor is moved outside the graphics window boundary, it is not visible since the window

desktop environment is drawn to the overlay planes, which have display priority over the cursor plane.

Typically, the user does not need to directly read or write pixels in the frame buffer. However, for those applications which require direct access, Starbase provides the gescape function R_GET_FRAME_BUFFER, which returns the virtual memory address of the beginning of the frame buffer. This gescape is discussed in the appendix of this manual. Frame buffer locations are then addressed relative to the returned address. The first byte of the frame buffer (byte 0) represents the upper left corner pixel of the screen. Byte 1 is immediately to its right. Byte 1279 is the last (right-most) pixel on the top line. The next 768 bytes of the frame buffer are not displayable. Byte 2048 is the first (left-most) pixel on the second line from the top. The last (lower right corner) pixel on the screen is byte number 2,096,383.

If more than one frame buffer bank is installed, bank switching must be used to access the additional memory. A number of Starbase calls may set the bank register so it is advisable to call bank_switch just prior to making accesses to the frame buffer pointer to ensure desired results.

If you are attempting to access the hardware directly while other processes are also using it (such as Starbase programs or window systems), you must obey semaphore protocols and save/restore any hardware registers you alter. See the description of the LOCK_DEVICE gescape for details on semaphore protocol.

The off-screen portion of the frame buffer may be accessed via the gescape procedure R_FULL_FRAME_BUFFER, documented in the appendix of this manual. Use this gescape carefully since other processes, Starbase, HP Windows/9000, and the X Window system, access the frame buffer off-screen memory.

Setting Up the Device On Series 300

The HP 98731 Device Driver can be used with the graphics display configured in either internal or external DIO-I address space, or in DIO-II address space. Refer to the *Configuration Reference Manual* for a description of internal and external DIO-I address space and DIO-II address space.

Note If the HP 98730 is configured as an external display, there will *not* be an Internal Terminal Emulator (ITE) for that device. Since it is the ITE that normally initializes the display, external devices must be reset after power-up by running a simple Starbase program with a mode of `RESET_DEVICE` in the `gopen` call. It may also be necessary to run this program after running an application which manipulated the overlay color map, such as a windows application program. An example program which could be called from `/etc/rc` during power-up is given at the end of this section. For more details concerning the effects of `RESET_DEVICE`, see the "Device Initialization" information in this section.

The Graphics Interface card may be installed in any DIO slot in the computer's backplane or in any I/O slot of the expander.

DIO-I Switch Settings

The graphics interface card has a single 8-bit address select switch. Looking at the switches so that the dot is in the lower left corner, the left-most switch is labeled DIO1 to the bottom (0), and DIO2 to the top (1). To configure the system in DIO-I space, this switch must be set to the DIO1 (0) position. The next switch to the right is labeled INT and determines if the HP 98730 workstation is configured as an internal or external display. In addition, the next six switches to the right are labeled for select code determination (five of the six switches are actually used for the select code). There is also a jumper labeled JP1.

The frame buffer uses two megabytes of I/O address space, starting at FB_BASE. The jumper (JP1) determines the address of FB_BASE.

JP1 is set to $200K FB_BASE address is $200 000

JP1 is set to $800K FB_BASE address is $800 000

Systems which use the HP 98730 display as a DIO-I system console will map the frame buffer to $200 000; systems which use the display as an external DIO-I device will map the frame buffer to $800 000.

The control space requires 128 Kbytes of space, starting at CTL_BASE. The six switches labeled "SC" determines the address of CTL_BASE. The HP 98730 may be configured as an external display, or as an internal display. Since only 64 Kbytes of space is normally allotted for external I/O select codes, two consecutive select codes will be used when configuring the device as an external display.

The following table lists the binary switch settings with the corresponding values of CTL_BASE for external I/O settings. The table also lists the select codes that are used for each setting.

Table HP98731-1. DIO-I Control Space Settings (External I/O)

Switch Setting MSB to LSB	CTL_BASE	DIO-I Select Code
01101010	$6A0000	10–11
01101100	$6C0000	12–13
01101110	$6E0000	14–15
01110000	$700000	16–17
01110010	$720000	18–19
01110100	$740000	20–21
01110110	$760000	22–23
01111000	$780000	24–25
01111010	$7A0000	26–27
01111100	$7C0000	28–29
01111110	$7E0000	30–31

For a system console (internal) the switch setting is 01010110 and the CTL_BASE is $560 000.

If the HP 98730 is configured as the system console, the CTL_BASE needs to be placed at $560 000 and the JP1 must be open (no jumper—or jumper is on one pin), which is an interal I/O setting. If the device is not used as the system console, the control space should not be placed in internal I/O space. It is likely to overlap the address space of other system hardware. In this case, an external I/O space setting should be selected with two consecutive select codes which are not used by the system.

DIO-II Switch Settings

If the left-most switch is set to DIO2 (1), the HP 98730 device can be used in DIO-II address space. In this mode, the next seven switches determine the DIO-II select codes to be used. An HP 98730 device will use three DIO-II select codes. Both the frame buffer and control space reside in the select code areas, so the jumper JP1 is ignored.

The control space requires 4 Mbytes of space, starting at CTL_BASE. The seven switches labeled "SC" at the top of the select switch determine the address of CTL_BASE. The frame buffer requires eight Mbytes of space, starting at FB_BASE.

Table HP98731-2. DIO-II Control Space Settings

Switch Setting MSB to LSB	CTL_BASE	DIO-II Select Code	FB_BASE
10000101	$01400000	133	$01800000
10001001	$02400000	137	$02800000
10001101	$03400000	141	$03800000
10010001	$04400000	145	$04800000
10010101	$05400000	149	$05800000
10011001	$06400000	153	$06800000
10011101	$07400000	157	$07800000
10100001	$08400000	161	$08800000
10100101	$09400000	165	$09800000
10101001	$0A400000	169	$0A800000
10101101	$0B400000	173	$0B800000
10110001	$0C400000	177	$0C800000
10110101	$0D400000	181	$0D800000
10111001	$0E400000	185	$0E800000
10111101	$0F400000	189	$0F800000
11000001	$10400000	193	$10800000
11000101	$11400000	197	$11800000
11001001	$12400000	201	$12800000
11001101	$13400000	205	$13800000
11010001	$14400000	209	$14800000
11010101	$15400000	213	$15800000
11011001	$16400000	217	$16800000
11011101	$17400000	221	$17800000
11100001	$18400000	225	$18800000
11100101	$19400000	229	$19800000
11101001	$1A400000	233	$1A800000
11101101	$1B400000	237	$1B800000
11110001	$1C400000	241	$1C800000
11110101	$1D400000	245	$1D800000
11111001	$1E400000	249	$1E800000
11111101	$1F400000	253	$1F800000

DIO-II displays may be used as the system console or as external displays. In order to use the display as system console, it must be configured as the first DIO-

II display in the system, and there must be no DIO-I console or remote terminals. Being the first DIO-II device means that it has the lowest DIO-II select code in the system. In order to use a HP 98730 device as a DIO-II system console, select code 133 is recommended.

Note

It is necessary to increase some of the HP-UX tunable system parameters due to the size of the DIO-II mapping of an HP 98730 device. For details on how to reconfigure your kernel, refer to the *HP-UX System Administrator Manual* (particularly the "Configuring HP-UX" section in "The System Administrators Toolbox" and the "System Parameters" appendix.

It is essential that you consult the above referenced HP-UX documentation *before* you attempt to reconfigure your system. It is possible to adversely affect your HP-UX system if a mistake is made. Ensure you have an understanding of these procedures before proceding.

In order to run an HP 98730 device in DIO-II the following system parameters must have these minimum values:

```
shmmax        0xC00000
shmmaxaddr    0x1800000
   dmmin      16
   dmmax      1024
   dmtext     1024
   dmshm      1024
```

Example Program to Reset the HP 98730

```
/*
 * Starbase program: reset98730.c
 * Compile: cc -o reset98730 reset98730.c -ldd98730 -lsb1 -lsb2
 * Destination: /usr/bin
 * Execute: add line to the /etc/rc - "/usr/bin/reset98730 /dev/crt.external"
 *
 * Example program to be put in /etc/rc for resetting an external HP 98730
 *  device during power-up.
 */
#include <starbase.c.h>

main(argc,argv)
int argc; char *argv[];
{
   int fildes;

   if ((fildes = gopen(argv[1],OUTDEV,"hp98730",INIT|RESET_DEVICE)) < 0)
      printf("External HP 98730 %s initialization failed.\n",argv[1]);
   else {
      printf("External HP 98730 %s initialization succeeded.\n",argv[1]);
      gclose(fildes);
   }
}
```

Setting Up the Device On the Series 800

Up to four HP 98730 devices can be connected to a Series 800 SPU using four A1017A interface cards. However, it is recommended that only two HP 98730 devices have the Internal Terminal Emulator (ITE) or window systems running on them. With the A1047A interface, only two devices can be connected, both of which may run an ITE.

The Series 800 ITE supports power-fail recovery on the HP 98731 device, but Starbase does not support the feature. If you want to support power fail, you must catch the power-fail signal and save any Starbase state needed. Then, `gclose` the device and `gopen` the device again when the power turns on.

Address Space Usage On Series 300

The HP 98730 device is memory mapped into a processes virtual address space, starting at the value specified by the environment variable `SB_DISPLAY_ADDR`. If this variable is not set, mapping defaults to `0xB00000`. The control space starts at this address and grows towards larger address values. After the control space comes the frame buffer and then shared memory mapped for Starbase drivers. The size of the address space used for control space and the frame buffer depends on whether the device is used in DIO-I or DIO-II. In DIO-I, control space consumes 128 Kbytes and the frame buffer uses 2 Mbytes. In DIO-II, control space is 4 Mbytes and the frame buffer is 8 Mbytes. The size of the Starbase drivers' shared memory is always the same and slightly less than 300 Kbytes.

If your application maps memory pages to specific addresses, or needs a large stack, you may need to adjust `SB_DISPLAY_ADDR` to avoid conflicts.

Special Device Files (mknod) On Series 300

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 `crt`.

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

When the device is at the internal DIO-I address (refer to the "Switch Settings" section) the `mknod` parameters should create a character special device with a major number of 12 and a minor number of 0.

```
mknod /dev/crt c 12 0x000000
```

When the device is at an external DIO-I or any DIO-II address (refer to the "Switch Settings" section) the `mknod` parameters should create a character special device with a major number of 12 and a minor number of 0x⟨*sc*⟩0200 where ⟨*sc*⟩ is the two-digit external select code in hexadecimal notation.

```
mknod /dev/crt c 12 0x⟨sc⟩0200
```

The HP 98730 driver may also be used for the overlay planes in graphics mode. The minor number may be set to cause Starbase drivers to use either three or four overlay planes. When running to three planes, one plane is still reserved for cursors. When running to all four overlays, only the hardware cursor is available for Starbase graphics echoes. If more than one echo is requested, or if another process is using the cursor, the request for another echo will fail. Note that since the terminal emulator and window system operate in the overlay planes also, there will be interactions with these processes if a graphics driver is opened in this manner while these processes are present. To open the HP 98731 Device Driver to three overlay planes instead of the graphics planes, the last byte of the

minor number must be 1. To run to all four overlays, the last byte of the minor number must be three.

For example, when the device is at an internal DIO-I address, the `mknod` parameters for the overlay device, with one plane reserved for cursors, should create a character special device with a major number of 12 and a minor number of 1.

```
mknod /dev/ocrt c 12 0x000001
```

To create a device file for all four overlays, the command would be:

```
mknod /dev/o4crt c 0x000003
```

When the device is at an external DIO-I address or any DIO-II address (refer to the section on "Switch Settings") the `mknod` parameters for the same device should create a character special device with a major number of 12 and a minor number of 0x⟨*sc*⟩0201 or 0x⟨*sc*⟩0203 where ⟨*sc*⟩ is the two-digit select code.

```
mknod /dev/ocrt c 12 0x⟨sc⟩0201
```

Special Device Files (mknod) On the Series 800

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 details. 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 files, however the names that are suggested for the devices are `crt`, `crt0`, `crt1`, or `crt2`.

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

When creating the device file the `mknod` parameters should create a character special device with a major number of 14 and a minor number of the following format (where ⟨*lu*⟩ is the two-digit hardware logical unit number):

```
mknod /dev/crtx c 14 0x00⟨lu⟩00
```

The HP 98730 Device Driver may also be opened to the overlay planes in graphics mode. If the last byte of the minor number is one, three overlay planes are used for graphics (and the fourth plane is reserved for cursors for processes running in the image planes). If the last byte of the minor number is three, four overlay planes are used for graphics. Since the ITE and window system operate in the overlay planes also, there will be interactions with these processes if a graphics driver is opened in this manner while these processes are present.

To open all four overlay planes, the `mknod` parameters should create a character special device with a major number of 14 and a minor number of three.

```
mknod /dev/ocrt4 c 14 0x00⟨lu⟩03
```

To open three overlay planes, the `mknod` parameters should create a character special device with a major number of 14 and a minor number of one.

```
mknod /dev/ocrtx c 14 0x00⟨lu⟩01
```

Linking the Driver

The HP 98731 Device Driver is the file named `libdd98731.a` in the `/usr/lib` directory. This device driver may be linked to a program using the absolute path name `/usr/lib/libdd98731.a`, an appropriate relative path name, or by using the `-l` option `-ldd98731`. This driver also requires the math library to be linked with C programs. For example, to compile and link a program for use with this driver, use:

```
cc example.c -ldd98731 -lsb1 -lsb2 lm -o example
fc example.f -ldd98731 -lsb1 -lsb2 -o example
pc example.p -ldd98731 -lsb1 -lsb2 -o example
```

depending upon the language being used.

Usage and Restrictions

When a device file for the overlay planes is used at **gopen** time, bank switching, shading, and depth cueing are not supported.

Graphics applications that want to talk to an HP Windows/9000 graphics window may use this driver, if the window was created with the `-N` option (this designates the window as an `IMAGE` graphics window). Refer to the HP Windows/9000 documentation for more details.

Graphics applications that want to open a local X window may use this driver if the window is in the image planes.

Up to 32 HP 98731 device drivers may be opened to the same device simultaneously from any combination of one or more processes.

HP Windows/9000 See-Thru Color

This device driver ignores the `SB_OV_SEE_THRU_INDEX`. For more details on its usage refer to the "HP 98730 Device Driver" section.

Cursors

The HP 98731 Device Driver implements cursors using either the hardware cursor or overlayed software cursors. If no processes have opened all four overlay planes, the fourth overlay plane is used for overlayed software cursors either by the HP 98730 or the HP 98731 drivers.

You can think of the fourth overlay plane used for cursors as a separate "cursor plane". Any data in the cursor plane will be displayed over data in the graphics planes. Data in the other three overlay planes will be displayed over data in the graphics planes and the cursor plane. For example, suppose a graphics application is running in the graphics planes while the window manager is running in three of the overlay planes. If the application has a Starbase cursor in the overlay cursor plane, the cursor will always be visible inside regions of see-thru because the cursor has display priority over the graphics. If the cursor is moved outside of regions of see-thru then it is not visible since the non-see-thru regions in the overlay planes have display priority over the cursor plane.

The HP 98730 Display Station also supports a hardware cursor that supports all Starbase echo types. The hardware cursor is drawn to a fifth and sixth overlay plane accessible only by the hardware cursor. There is only one hardware cursor available. Usage of the hardware cursor is defined as follows:

1. If an application is running in a Starbase environment only (that is, neither Windows/9000 nor X Windows is not running), the hardware cursor is given to the first process that attempts to use cursors.
2. The X Window system always uses the hardware cursor for the X sprite.
3. By default, if Windows/9000 is active, the window system gets usage of the hardware cursor. The user may prevent the window system from using the hardware cursor by setting the `WMCONFIG` environment variable appropriately. See the HP Windows/9000 documentation for details.
4. Via the `gescape R_ECHO_CONTROL`, there is a mechanism for the user to control usage of the hardware cursor. This `gescape` is discussed in the appendix.

If the hardware cursor is already being used by another process, overlayed software cursors are used by the HP 98731 driver. If the fourth overlay plane is not available for cursors, an error will be generated when any attempts are made to turn on the cursors. In an X window, cursors may be available even when the fourth overlay plane is not. See *Starbase Programming with X11* for more information.

If a users application never uses cursors, the driver will never attempt to allocate the hardware cursor. However, once the driver has allocated the hardware cursor, the driver does not relinquish usage of the hardware cursor until `gclose` time.

If allocation of the hardware cursor was not successful, resources for the software cursor area are allocated (that is, offscreen areas for raster echo definitions). Once resources for software cursors have been allocated, the driver always uses software cursors and never again attempts to use the hardware cursor.

The following functions will cause the driver to attempt to allocate cursor resources (that is, either the hardware cursor or software cursor resources):

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

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/crt`.)

Kind — This indicates the I/O characteristics of the device. This parameter may be one of the following:

- `INDEV`—input only.
- `OUTDEV`—output only.
- `OUTINDEV`—input and output.

Input may be done with this driver only when opened to an HP Windows/9000 image window or an X Window system window.

Driver — This is the character representation of the driver type. This must be `hp98731`. For example:

`"hp98731"`	*for C.*
`'hp98731'//char(0)`	*for FORTRAN77.*
`'hp98731'`	*for Pascal.*

Mode — The mode control word consistng of several flag bits which are *or* ed together. Listed below are the flag bits which have device-dependent actions. Those flags not discussed below operate as defined by the gopen procedure.

- `SPOOLED`—cannot spool raster devices.
- `0`—open the device, but do nothing else. This will set the color map mode to `CMAP_NORMAL` but will not initialize the color map itself. This will also disable `blending` if it was left enabled. This mode will not affect pixel panning and zooming. In an X window the color map mode is initialized to be consistent with the X color map.
- `INIT`—open and initialize the device as follows:
  1. Clear frame buffer to 0s.
  2. Reset the color map to its default values.

3. Enable the display for reading and writing.
4. Initialize the transform engine's microcode.
5. Download the transform engine's microcode (if it has not already been done).
6. Restore pixel pan and zoom hardware for normal viewing, if opened to the image planes.
7. In an X window, a new color map is created, and the color map mode is initialized to be consistent with the X color map.

- `RESET_DEVICE`—open and reset the device as follows:
  1. Clear frame buffer and overlays to 0s.
  2. Reset the color map to its default values.
  3. Clear the overlay color map.
  4. Enable the display for reading and writing.
  5. Download the transform engine's microcode.
  6. Initialize the transform engine's microcode.
  7. Restore pixel pan and zoom hardware for normal viewing, if opened to the image planes.
  8. In an X window, a new color map is created, and the color map mode is initialized to be consistent with the X color map.

  Note that the `RESET_DEVICE` flag bit should be used with caution: it will adversely affect any other processes using the device. This flag bit is intended to reset a device completely: this should only be necessary for devices in an unknown state, such as a device powered up in an external I/O space. Most programs should not use this flag bit.

Syntax Examples

To open and initialize an HP 98731 device for output:

For C Programs:

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

For FORTRAN77 Programs:

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

For Pascal Programs:

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

Special Device Characteristics

For device coordinate operations, location $(0,0)$ is the upper-left corner of the screen with X-axis values increasing to the right and Y-axis values increasing down. The lower-right corner of the display is therefore $(1279, 1023)$.

Offscreen Memory Usage

Offscreen memory is managed by a global resource manager to insure that multiple processes do not step on each other when using the offscreen. Offscreen is used by the device driver for:

- polygon fill patterns
- raster echo definitions (if software cursors are used)

The offscreen memory is not allocated for any of the above functions unless the function is used. Therefore, if an application never does filled polygons, and never uses software cursors, the driver does not use the offscreen memory.

Refer to the `gescape R_OFFSCREEN_ALLOC` for information on using the offscreen areas for personal use.

Device Defaults

Number of Color Planes

When the `gopen` procedure is called, this driver asks the device for the number of color planes available. This number can be either 3 or 4 (in overlays), 8, 16, or 24. The device driver then acts accordingly.

Dither Default

The default number of colors searched for in a dither cell is 2. The number of colors allowed in a dither cell is 1, 2, 4, 8 or 16.

Raster Echo Default

The default raster echo is the 8×8 array:

255	255	255	255	0	0	0	0
255	255	0	0	0	0	0	0
255	0	255	0	0	0	0	0
255	0	0	255	0	0	0	0
0	0	0	0	255	0	0	0
0	0	0	0	0	255	0	0
0	0	0	0	0	0	255	0
0	0	0	0	0	0	0	255

The maximum size allowed for a raster echo is 64×64 pixels.

By default, all echo types are written using the dedicated hardware cursor. If the hardware cursor is not available, cursors are written in the fourth overlay planes. If no overlay plane is reserved, cursors are not available. In an X window, cursors may be available even when the cursor plane is reserved. See the *Starbase Programming with X11* manual for more information.

Color Planes Defaults

The default configuration is an 8-plane color mapped system regardless of the number of frame buffer banks installed.

All planes in the first bank are display enabled. All planes in the first bank are write enabled.

Semaphore Default

Semaphore operations are enabled.

Line Type Defaults

The default line types are created with the bit patterns shown below:

Table HP98731-3.

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

In addition, a point plot line type is supported. This is accessed by using a line type of -1. This line type will cause one pixel to be plotted at each vertex of the line segment.

Default Color Map

If the fourth `gopen` parameter is zero (`0`), the current hardware or X window system color map is used on color displays.

If the fourth `gopen` parameter is `INIT`, the current color map is initialized to the default values shown in the following table.

Table HP98731-4. 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

Use the `inquire_color_table` procedure to see the rest of the 255 colors.

When `INIT` is used in the `shade_mode` procedure call the color map initialization is based on the value of the mode parameter and on the number of frame buffer banks installed.

`mode=CMAP_NORMAL` Same as the table above. Only one bank can be displayed at a time, unless video blending is enabled.

`mode=CMAP_MONOTONIC` The color map will be initialized as:

```
for (i=0; i<256; i++) {
  cmap[i].red = cmap[i].green = cmap[i].blue = i/255.0;
}
```

Only one bank can be displayed at a time, unless video blending is enabled.

`mode=CMAP_FULL`

With less than three banks available, the color map will be initialized as three bits red, three bits green, and two bits blue. The three most significant bits are red, and the two least significant bits are blue. Only one bank of the first two can be displayed at a time.

Note: This driver requires at least eight planes in `CMAP_FULL` mode, or at least 16 planes if double buffered.

With three or more banks available, the color map will be initialized as the `CMAP_MONOTONIC` case above but now the first bank of eight will go through the blue portion of the color map, the second bank goes through the green portion, and the third bank goes through the red portion. In this mode the color map is transparent and the eight bits from each bank drive the appropriate DAC. The color map could be subsequently modified in this mode to perform functions such as gamma correction or double buffering of 4-bits per color.

Red, Green and Blue

With a raw device or HP Windows/9000, each file descriptor opened as an output device has a color table associated with it. If multiple file descriptors are open to the same device, the color table and the device's color map may not always be identical. The color table does not track the color map if the device's color map is changed by another file descriptor path.

For Starbase procedures that have parameters for red, green and blue, the way the actual color is chosen depends on the current `shade_mode` setting.

`mode=CMAP_NORMAL`

The color map is searched for the color that is closest in RGB space to the one requested. That color map index is written to the frame buffer for subsequent output primitives. It is more efficient to select a color with an index rather than specifying a color with red, blue, and green values in this mode because it takes extra time to figure out which index in the color table most closely matches the specified color.

`mode=CMAP_MONOTONIC` The red, green, and blue value is converted to an intensity value using the equation:

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

This intensity is converted to an index value by mapping intensity 0.0 to the minimum index set be `shade_range` and intensity 1.0 to the maximum index set by `shade_range`. This mode is useful for displaying a high quality monochrome picture on an 8-plane system from data that produces a high quality color picture on a 24-plane system.

`mode=CMAP_FULL` With less than three banks installed, the color is converted to a color map index by the equation:

```
index=(round(red*32767)>>7) & 0xE0 |
(round(green*32767)>>10) & 0x1C |
(round(blue*32767)>>13)
```

This equation will be used in this mode regardless of whether the user has modified the color map.

With three or more banks installed, the red, green and blue values are each multiplied by 32,767, shifted right seven places, then written to the appropriate bank.

Note When using an X window, some color map sharing between **gopen** functions is possible. See the *Starbase Programming with X11* manual for more information.

Starbase Functionality

Exceptions to Standard Starbase Support

The following commands are supported under the listed conditions:

`inquire_fb_configuration`

An HP 98730 device running the HP 98731 Device Driver will report `image_banks` as 5 if the system has 24 display planes and the dedicated Z-buffer. If the dedicated Z-buffer is installed in this way, it is possible to access it with `block_write`, `block_read`, and `block_move`. The Z-buffer may be selected for read/write using `bank_switch`. The Z-buffer may not be displayed. The Z-buffer cannot be rendered to by the graphics accelerator. If less than 24 planes are installed, the presence of a Z-buffer will not be reported.

`interior_style`

If the polygon fill type is `INT_HATCH` then the following functionality will not work correctly:

- hidden surface removal.
- shading and lighting.
- depth cueing.
- backfacing attributes and culling.
- splines, quadrilateral meshes, and triangle strips will not be hatched.

Performance is also degraded in this mode.

`text_precision`

Only `STROKE_TEXT` precision is supported.

block_read, block_write

The `raw` parameter for the `block_read` and `block_write` commands is normally ignored by this device driver. To use the `raw` mode, you must call the `R_BIT_MODE` or `R_DMA_MODE gescapes` discussed in the appendix of this manual. If the `raw` parameter is `TRUE`, then no clipping will be done.

When running to a window, the window offsets from the upper left hand corner of the screen will be added to `block_write` and `block_read` start locations. If you do not want this offset added, you should subtract the offsets from your start point. These offsets can be computed by calling the `gescape` functions `R_GET_FRAME_BUFFER` and `R_GET_WINDOW_INFO`. Using the frame buffer pointers returned by these routines, the window offsets are:

```
y_offset=(window_ptr-fb_ptr)/2048

x_offset=(window_ptr-fb_ptr)-(y_offset*2048)
```

This would be useful, for example, if you wished to write a polygon fill pattern offscreen to a frame buffer absolute address while running in a window.

Fast Alpha and Font Manager

This device driver does *not* support raster text calls from the fast alpha and font manager library.

Parameters for gescape

The following `gescape` functions are common to two or more of the Hewlett-Packard displays supported by Starbase. Detailed information about these functions can be found in Appendix A.

- `BLINK_INDEX`—Alternate between HP 98730 hardware color maps.
- `BLINK_PLANES`—Blink display (blink rate is 3.75 Hz for this device.)
- `IMAGE_BLEND`—Enable/disable video blending.
- `LS_OVERFLOW_CONTROL`—Sets options for overflow situations.
- `OVERLAY_BLEND`—Control blending of overlay plane frame buffer.
- `PAN_AND_ZOOM`—Pixel pan and zoom.
- `PATTERN_FILL`—Fill polygon with stored pattern.

- R_BIT_MASK—Bit mask.
- R_BIT_MODE—Bit mode.
- R_DEF_ECHO_TRANS—Define raster echo transparency.
- R_DEF_FILL_PAT—Define fill pattern.
- R_DMA_MODE—Changes the definition of `raw` for block writes.
- R_ECHO_CONTROL—Control hardware cursor allocation.
- R_ECHO_FG_BG_COLORS—Define echo attributes.
- R_ECHO_MASK—Define cursor mask.
- R_FULL_FRAME_BUFFER—Full frame buffer.
- R_GET_FRAME_BUFFER—Read frame buffer address.
- R_GET_WINDOW_INFO—Returns frame buffer address of window.
- R_LINE_TYPE—Define line style and repeat length.
- R_LOCK_DEVICE—Lock 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_TRANSPARENCY_INDEX—Specify HP 98720 transparency index.
- R_UNLOCK_DEVICE—Unlock device.
- READ_COLOR_MAP—Read color map.
- SET_BANK_CMAP—Set frame buffer bank color maps.
- SWITCH_SEMAPHORE—Semaphore control.
- TRANSPARENCY—Allows "screen door" for transparency pattern.
- ZWRITE_ENABLE—Allows creation of 3D cursors in overlay.

The following `gescape` functions are unique to this driver and are presented in the following section.

- CLIP_OVERFLOW—Change X Window system hierarchy.

- GAMMA_CORRECTION—Enable/disable gamma correction.
- POLYGON_TRANSPARENCY—Define front facing and backfacing polygon transparency patterns.

CLIP_OVERFLOW

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

This `gescape` allows the user 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 the user 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

```
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.

GAMMA_CORRECTION

The ⟨*op*⟩ parameter is GAMMA_CORRECTION.

This gescape allows the user to enable or disable GAMMA_CORRECTION in the HP 98730 hardware.

The user passes 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

```
/* 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

```
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

```
{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);
```

POLYGON_TRANSPARENCY

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

This `gescape` allows the user to define separate "screen door" transparency patterns for frontfacing and backfacing polygons. The user may define patterns that disable writes to any pixels within a 4×4 cell. This cell is duplicated over the entire screen.

The user passes 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 the user, 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 HP98731-5.

15	...	2	1	0

Table HP98731-6.

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

```
/* 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

```
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

```
{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);
```

Performance Tips

General

1. As with any driver, buffering is done to enhance performance. If `buffer_mode` is turned off or if an inordinate amount of `make_picture_current` or `flush_buffer` calls are done, performance can be degraded substantially.
2. Performance optimizations have been made so that sequential calls of the same output primitive with no intervening attribute changes or different primitive calls goes faster. For example, the sequence `polygon, polygon, polyline, polyline` is faster than `polygon, polyline, polygon, polyline`. Also `line_color, polyline, polyline` is faster than `line_color, polyline, line_color, polyline`. So grouping by primitive and subgrouping primitives by attribute can give substantial performance improvements.
3. Typically, the HP 98731 rendering engine renders primitives from its internal buffer as the system CPU is doing other things. Substantial performance benefits can be realized from this parallel processing.

   However, certain operations will cause the CPU to wait for the HP 98731 to finish emptying its buffer. An example of this wait is the `make_picture_current` operation. Also, any operation that reads information from the HP 98731 may cause this wait to occur. Two operations read the matrix values from the HP 98731: `pop_matrix2d` and `pop_matrix3d`. If the values in the popped matrix are not needed, use `pop_matrix`, which does not cause any information to be read from the HP 98731. Also, `block_read` and `block_write` will also cause the driver to use it.

Screen Clears

1. Screen clears will be significantly faster if the area to be cleared starts on a 128-pixel boundary and is some multiple of 128 pixels wide. This can be checked by using the Starbase routines `transform_point` and `vdc_to_dc`

to convert the bounds of the clear rectangle to device coordinates. Screen clears to the default `vdc_extent` will be aligned.

2. For programs which use zbuffer hidden surface removal with the dedicated zbuffer, it is much faster to clear the zbuffer simultaneously with screen clears than to do the clears sequentially. This is accomplished by calling `clear_control` with `CLEAR_ZBUFFER` *or*ed into the mode word. When this is done, subsequent calls to `clear_view_surface` and `dbuffer_switch` will cause the zbuffer to be cleared also. See the manual page for `clear_control` for more details.

Rendering

1. When doing shaded polygons, the fewer the features, the faster the polygon generation. Positional viewpoint and light sources can significantly degrade performance.

2. With shading, and Z-buffering off, the HP 98731 rendering engine runs at full speed, when rendering flat shaded polygons. These two rendering techniques slow the rendering of polygons on the HP 98731. This is especially noticeable on large polygons. Turning on any one of these could noticeably lower the rendering performance.

   Using the pattern `gescape` or replacement rules that require extra reads of the frame buffer (e.g. source *or* destination) will also degrade performance. It takes time to do the extra reads.

3. Rendering mode commands such as `hidden_surface`, `shade_mode`, and `double_buffer` can be slow. These should not be unnecessarily called. For example, it is not necessary to repeatedly call `hidden_surface` from an animation loop; it is intended that these routines be called to initialize a rendering mode and are only called again to change it.

Raster Operations

1. The procedure `block_move` is faster than `block_read` or `block_write` since the hardware frame buffer block mover can be used.

2. The performance of `block_read` and `block_write` is significantly better if both the source and destination begin on the same byte boundary, since

data can be transferred 32-bits at a time rather than one byte at a time. For example, one way to ensure this condition is to define pixel arrays as type short (16-bit integers) and then start `block_read` and `block_write` actions on even pixels only. This can more than double performance. Note that the bite boundaries are relative to the screen address, not the window address.

3. `block_write` on Series 800 machines with the A1047A interface can go faster by using DMA. See `R_DMA_MODE gescape`.

Cautions

The following cautions are provided in using this driver:

1. As mentioned previously, accessing the off-screen portion of the frame buffer (using `gescape` functions) should be done with care, since other processes access this region. The overlay off-screen contains the ITE font (which is regenerated when control-shift-reset is done on the ITE keyboard) and may contain any number of window systems fonts depending on the current window usage.

2. Polygons of up to 255 vertices (after clipping) are supported. If a polygon has more than 255 vertices, only the first 255 vertices are displayed.

3. Certain `gescape` functions should be used with caution since they bypass protection mechanisms used to prevent multiple processes from interfering with each other. For example, since the hardware resources can only be rationally used by one graphics process at a time, the driver activates a semaphore and locks the device before doing any output. This ensures, for example, that process A will not change the replacement rule while process B is in the middle of filling a polygon. It also prevents the terminal (`tty`) driver from overwriting any graphics processes that are outputting to the device. The driver unlocks the device when done processing output. Some of the `gescape` functions listed in this chapter allow the user to change this locking mechanism and should be used with *great caution.*

4. When using the HP 98730 device with a graphics accelerator it is possible for illegal operations to cause the transform engine or scan converter hardware to enter an unknown state. If this happens, Starbase will report an error the next time it tries to use the hardware. The user will see this as a `Transform engine timed out` or `Hardware/scan_converter time out` error. These are Starbase errors 14 and 52 respectively. This is a very serious error condition. If the HP 98731 Device Driver is being used, this is a fatal error. When this error is discovered, Starbase reports the error and aborts execution.

   If an application needs to take some emergency action before an untimely termination, such as saving valuable data, the application should check for these error conditions and take appropriate measures. Errors may be

caught by an application using the `gerr_control` procedure described in the *Starbase Reference* manual.

It is also possible to avoid the termination completely if the application's error handler does not return control to Starbase. It is, however, impossible to proceed with any graphics efforts using the accelerator until it is reset.

Opening Windows

The HP 98731 accelerated driver can open a number of windows in the image planes. The limits placed on these windows are:

1. The HP 98731 driver supports up to 31 accelerated windows operating simultaneously. Furthermore, it permits an accelerated window to be obscured by, at most, 31 other rectangles (for example, corners of windows).
2. When an image plane window is rendered to by the accelerator and is obscured by more than 31 rectangles, rendering is halted until that window has moved up enough in the window stack to be obscured by fewer than 31 rectangles. It is possible for a program to detect when this occurs by passing a procedure address to the Starbase `gescape` procedure with opcode `CLIP_OVERFLOW`. This procedure is then called whenever the clip list overflows. Refer to the HP 98730 chapter in this manual for information on this `gescape` opcode.
3. When a window is about to become obscured by more than 31 windows and the accelerator hardware is currently rendering to that window, the window system is locked until the accelerator is finished with the current set of primitives. The calling process will become blocked and the `CLIP_OVERFLOW` procedure will be called by Starbase.

The above guidelines only apply to windows in the image planes. For example, in combined mode, overlay plane windows which overlap image plane windows do not count against the limit of 31 obscuring rectangles. The limit only applies to image-plane windows which overlap other image-plane windows. We recommend that non-graphical windows (for example, terminal emulator windows) and

graphical windows that don't need to use the graphics accelerator be placed in the overlay planes.

Note that accelerated overlay windows are not supported with the HP 98731 driver.

The Hardware Cursor

The HP 98731 color map supports a single, independent hardware raster or vector cursor. The hardware cursor is a 64×64×2 bit raster pattern that is conceptually in front of the overlay planes. It is defined with a 64×64 bit/pixel color pattern and a 64×64 bit/pixel transparency pattern. When the X11 server is started, it uses the hardware cursor for the window cursor.

As with the overlay planes, one of the colors is a transparency color used to see through to the overlay and image planes. This means that a raster cursor can have no more then two significant colors (one additional color is used for the transparency pattern). The two colors used by the cursor are based on 24-bit RGB values and are independent of the other color maps.

When the X11 server is using the hardware cursor and a program defines a Starbase echo in an image window, the echo is placed by default in the cursor plane. When a cursor plane is not available, the HP 98730 driver renders the cursor in the image planes. The echo colors will be chosen from the color map associated with that window. When it is an image plane window, the X standard color map is used. This means that when an image plane window is the focused window, the X standard color map will be loaded into the overlay plane hardware color map.

Z Buffer

For graphics operations that require a Z buffer such as hidden-surface removal, a dedicated Z buffer board must be installed in the HP 98731. When the Z buffer board is installed and an accelerated image-plane X11 window is opened, the X11 server also associates a corresponding portion of the Z buffer with the window. This Z-buffer allocation is automatically moved and resized as the window is moved and resized. It is also obscured by other windows in the image planes.

Contents

SMD

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

`SMDpixel` and `SMpixel3` is the 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 `-lddSMDpix`. For example, to compile and link a program for use with this driver use:

```
cc example.c -lddSMDpix -lsb1 -lsb2 -o example
fc example.f -lddSMDpix -lsb1 -lsb2 -o example
pc example.p -lddSMDpix -lsb1 -lsb2 -o example
```

Which example you use depends on the language being used.

`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 as the following examples shows:

```
cc example.c -lfontm -lddSMDpix -lsb1 -lsb2 -o example
cc example2.2 -lfontm -lddSMDpln -lsb1 -lsb2 -o example2
```

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`.

Device Defaults

Number of Color Planes

You can specify the number of planes of depth for the memory buffer if using `SMDpixel` or `SMDplane`. The valid values for `SMDpixel` are 1, 3, 4, 6, or 8. The valid values for `SMDplane` are 1, 3, 4, 6, 8, 16, or 24. The default depth for `SMDpixel` is 8 planes. The default depth for `SMDplane` is 1 plane. For `SMDpixel3`, the depth is fixed at 24. The device driver then acts accordingly.

Dither Default

The default number of colors searched for in a dither cell is 2. The number of colors allowed in a dither cell is 1, 2, 4, 8 or 16. For devices having 24 or more planes in `CMAP_FULL` mode (see `shade_mode`) dithering is not supported since full 24-bit color is available. If you are double-buffering with 12 planes per buffer, the number of colors allowed in a dither cell is 1, 2, or 4.

Semaphore Default

There are no semaphore operations supported on a memory buffer.

Line Type Defaults

The default line types are created with the bit patterns shown in the following table:

Table SMD–1.

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

Default Color Map

The number of entries in the color map is is the number of planes in the memory buffer. The color map, as far as it goes, is initialized to the default values shown in the following table.

Table SMD–2. 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

Use the `inquire_color_table` procedure to see the rest of the 255 colors.

When INIT is used in the shade_mode procedure call, the color map initialization is based on the value of the mode parameter and the number of frame buffer banks requested.

mode = CMAP_NORMAL	Same as the previous table.

mode = CMAP_MONOTONIC	The color map is initialized as:
	`for (I = 0; I < 256; I++)`
	`cmap[I].red = cmap[I].green = cmap[I].blue = I/255.0;`

mode = CMAP_FULL	With 24 planes specified, the color map is initialized as the CMAP_MONOTONIC case above; but now the first bank of 8 goes through the blue portion of the color map, the second bank goes through the green portion, and the third bank goes through the red portion. The color map could be subsequently modified in this mode to perform functions like gamma correction or double buffering of 4 bits per color.

Red, Green, and Blue

For Starbase procedures that have parameters for red, green, and blue, the way the actual color is chosen depends on the current shade_mode setting.

mode = CMAP_NORMAL	The color map is searched for the color that is closest in RGB space to the one requested. That color map index is written to the frame buffer for subsequent output primitives. It is more efficient to select a color with an index rather than specifying a color with red, blue, and green values in this mode because it takes extra time to figure out which index in the color table most closely matches the specified color.

`mode = CMAP_MONOTONIC` The red, green, and blue value is converted to an intensity value using the equation:

$0.30 \times red + 0.59 \times green + 0.11 \times blue$

This intensity is converted to an index value by mapping intensity 0.0 to the minimum index set by **shade_range**, and intensity 1.0 to the maximum index set by **shade_range**. This mode is useful for displaying a high-quality monochrome picture on an 8-plane system from data that produces a high quality color picture on a 24-plane system.

`mode = CMAP_FULL` The color values are mapped directly to an index with the assumption that the color map is set up to a predefined full color state.

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 HP 98720 transform engine driver are not supported. Explicitly, the functions not supported are:

```
backface_control
bf_control
bf_fill_color
bf_interior_style
bf_perimeter_color
bf_perimeter_repeat_length
bf_perimeter_type
bf_surface_coefficients
bf_surface_model
define_trimming_curve
depth_cue
depth_cue_color
depth_cue_range
hidden_surface
intline_width
interior_style (INT_OUTLINE)
interior_style (INT_POINT)
light_ambient
light_attenuation
light_model
light_switch
line_endpoint
shade_range
surface_coefficients
surface_model
viewpoint
zbuffer_switch
```

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.

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).

- Windows/9000 cannot run in an SMD memory buffer. However, SMD supports the capabilities of the "Font Manager Library" section of *Fast Alpha/Font Manager's Programmer's Manual.*
- 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's 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^{16}-1 (65,535).
- 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:

  - If Windows/9000 is being run with the application, either eliminate it from the application, or modify the windows environment to use less shared memory per application. See the *Windows/9000 Programmer's Manual,* "Programming Environment" for more information.

- 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

The following gescape functions are common to two or more devise drivers and discussed in the appendix 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 and are discussed here.

- 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

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` packing format and the **physical** depth for `SMDplane` (the physical depth always remains 8). If the user 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[1]).

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.

---

[1] 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 the user specifies 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`), the user 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 the user 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 the user may have done are not retained. This `gescape` works on memory buffers supplied by the user (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 $2^{16}-1$ (65,535) or less. This is because the vector generation algorithms can only handle up to 16 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 the user 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 the user specifies 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 the user 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 the user redefines the X, Y size too large to be allocated,

- failure is returned to the user 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 a user's 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, the user should call `gescape` with `SMD_ALLOCATE_MEMORY` to force allocation of the memory buffer. This insures that the SMD can allocate the required memory. The user 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 the user has no way of knowing how many bytes are available at allocation time.

SMD_SUPPLY_MEM_BUFF

The ⟨*op*⟩ parameter is SMD_SUPPLY_MEM_BUFF (supported by all SMD drivers).

This gescape allows the user 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), the user passed in a NULL pointer. If the error was 6 (IMPROPER_VALUE), the user tried to supply a memory buffer after the memory buffer had already been allocated.

SMD_GET_MEM_REQUIRED

The ⟨*op*⟩ 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_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.

Table of Contents

The Starbase-on-X11 Device Driver

Device Description

The Starbase-on-X11 (sox11) device driver library, `libddsox11.a`, allows 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 or 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

The name of the sox11 driver is `libddsox11.a`. This driver may be linked into your application using an absolute path name to the driver, or by using the `-l` option `-lddsox11`. 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`. For example, to compile and link a program for use with this driver, depending upon the source language used (C, Fortran, or Pascal):

```
cc example.c -o example  -lddsox11 -lsb1 -lsb2 -lXhp11 -lX11

fc example.f -o example  -lddsox11 -lsb1 -lsb2 -lXhp11 -lX11

pc example.p -o example  -lddsox11 -lsb1 -lsb2 -lXhp11 -lX11
```

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 `libXhp.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()`s 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 name of the driver type. The name of this driver is `sox11`. The exact string used for this argument depends on the programming language being used. 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\n", 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 the window is `gopen()`'d. 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 1. Default Line Types

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

Color Map Defaults

When an X11 window is gopen'd 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 HP Window Manager (hpwm), colormap focus can be set in three ways:

pointer	Anytime the pointer enters the window, that window is given the colormap focus.
explicit listener	Anytime the window receives a button click, it is given colormap focus.
tracked keyboard input	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	backface_control	bank_switch
bf_control	bf_fill_color	bf_interior_style
bf_perimeter_color	bf_perimeter_type	bf_perimeter_repeat_length
bf_surface_coeficients	bf_surface_model	dbuffer_switch
define_trimming_curve	depth_cue	depth_cue_color
depth_cue_range	hidden_surface	light_ambient
light_attenuation	interior_style (INT_OUTLINE, INT_POINT)	light_model
light_source	light_switch	shade_mode
shade_range	surface_coefficient	surface_model
viewpoint	zbuffer_switch	

Gescapes

The following gescapes are supported by the sox11 driver.

The following gescapes, which are common to many display drivers, are documented in appendix A:

- 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 for all keystrokes. Special function keys return keycodes as defined by the Xrlib routine, `XrInputMap(3X)`. This is the "cooked" mode for input keystrokes. This input model works correctly for HP-HIL language keyboards.

"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 (ie. 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

Window Resize

A widow 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 the window is `gopen()`'d 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.

Table of Contents

The X10 Windows Device Driver

Device Description

The Starbase X10 device driver library, `libddXn.a`, allows an application to utilize Starbase functions within the X10 window system. This driver implements the device dependent Starbase routines by calling the X10 Window library. This may be described as an implementation of Starbase "on top of " X.

This implementation allows Starbase applications to utilize the features of the X10 Window System. This includes running applications over the network, where the application runs as a client on one machine while using the X10 display server to perform I/O either on another machine or locally. This allows a Starbase application to use any HP 9000 machine running Starbase as its X10 client machine, while the same application may use any accessible hardware running an X10 server to perform all I/O operations. Note that the X10 server performing I/O for the application need not necessarily be running on HP equipment. Thus an X10 window serves as a virtual device for Starbase.

Setting Up a Starbase/X10 Environment

The Starbase/X10 driver can be used on any machine once Starbase and the X10 Window System have been installed. No special installation need be performed, and no special nodes need to be made in order to use the Starbase/X10 driver. As long as Starbase has been installed on the system, applications may run on any accessible display being controlled by an X10 server.

The Starbase/X10 driver also works in concert with HP Windows-X. If an application makes HP Windows-X calls, and the HP Windows-X environment has been started using the `wmstart` command, or alternatively the `wxstart` command, the Starbase/X10 driver will work using knowledge of the HP Windows-X design. Again no special nodes or installation need to be done in order to effect this functionality.

Linking the Starbase/Xn Driver with an Application

The name of this Starbase/X10 driver is `libddXn.a`. This driver may be linked into your application using an absolute path name to the driver, or by using the `-l` option `-lddXn`. The absolute path name of the driver is `/usr/lib/libddXn.a`. For example, to compile and link a program for use with this driver, depending upon the source language used ("C", Fortran, or Pascal):

```
cc example.c -o example  -lddXn -lsb1 -lsb2 -lXr -lX

fc example.f -o example  -lddXn -lsb1 -lsb2 -lXr -lX

pc example.p -o example  -lddXn -lsb1 -lsb2 -lXr -lX
```

Note that it is necessary to link in the `libXr.a` and `libX.a` libraries. Note also that `libXr.a` must be linked in before `libX.a`.

Programmatic Initialization

In order to use Starbase functionality within an X10 window, you must perform a `gopen()` call on the existing targeted X10 window. The window may be of any size and location on a device controlled by an X10 server. Multiple processes may `gopen()` the same window, and a single process may `gopen()` multiple windows.

In order to `gopen()` an X10 window, there must exist a file associated with the targeted window. The name of the file is then used as the first argument to the `gopen()` routine.

An X10 window to be used with the Starbase/X10 driver can easily be created using one of two methods. Firstly, the X10 window to be `gopen`'ed may be created using the HP Windows-X commands or the `wcreate_graphics()` routine. This provides source code compatibility with previous implementations of HP Windows 9000 and Starbase. Secondly, a Starbase application not utilizing HP Windows-X can call one of the utility routines provided `libXhp.a`, such as `XhpCreate()`. These utilities are provided as library routines within `libXhp.a`. For example, `XhpCreate()` serves to create an X10 window of specified size and location, on a specified display. Refer to `libXhp(3X)` for further details regarding `libXhp.a`.

Gopen Parameters

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

Path

The nature of the path name used as an argument to `gopen()` is dependent on the environment in which the application is running. Essentially an application can execute dependent on the HP Windows-X environment or an application can execute in a stand alone environment without needing HP Windows-X.

In the case where an application is using the HP Windows-X environment, the path name supplied to `gopen` is the path supplied to the `wcreate_graphics()` call. In this case the path name is effectively a link to a pty node. The Starbase/X10 driver later utilizes that path name to the pty node to acquire information about the window from HP Windows-X. Note that one can use the `wcreate` command, instead of the `wcreate_graphics()` routine.

When an application is executing without using the HP Windows-X environment, the path name may be supplied by the application or generated within one of the routines of `libXhp.a`. For example, if the X10 window was created using the `XhpCreate()` routine and no path name was supplied as an argument to `XhpCreate()`, a path name is generated by `XhpCreate()`. This path name should be used as an argument to `gopen()`. The path name returned by `XhpCreate()` is simply the name of a regular file whose contents represent the display name and windowId of the window to be `gopen`'d. The Starbase/X10 driver in this case reads the contents of the file to obtain information about the window to be `gopen`'d.

Note that the application could also supply a desired path name to the utility, `XhpCreate()`. In this case the path name supplied by the application to the `XhpCreate()` routine is also used as an argument to `gopen()`. After the call to `XhpCreate()` this file will contain the windowId and the display name of the window targeted to be `gopen()`'d.

Lastly there is nothing to prevent an application from using a utility of its own to create an X10 window to be `gopen`'d. If an application uses its own window creation scheme then the utility routine `XhpFile()`, found in `libXhp.a`, will prove useful in creating a file used as the argument to `gopen()`. Given a windowId and an optional display name `XhpFile()` will create a regular file containing the information needed during the `gopen()` call. The path name of this file, returned by `XhpFile()`, is used as the path

argument to `gopen()`. See `XhpFile(3x)` for details.

Kind — This indicates 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 are used then the X10 Window System will present a virtual input device interface where a keyboard is present as a CHOICE device and a three button mouse may be accessed as both a CHOICE device and a LOCATOR device.

Driver — This is the name of the driver type. The name of this driver is `Xn`. The exact string used for this argument depends on the programming language being used. For example:

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

Mode — This is the mode control word, which consists of several flag bits which are OR'd together. The RESET_DEVICE and INIT flags do **not** cause any hardware to be reset in the devices.

Gopen Examples

An X10 window must first exist before trying to use `gopen()` with X. The library routines in `libXhp.a` have been supplied to help prepare an X10 window ready to be `gopen()`'d. For example, the `XhpCreate()` routine creates an X10 window with the requested size and attributes on a specified display, and returns a path name to a file whose contents represent the display and windowID of the recently created X10 window.

One way for a C program to create and `gopen()` an X10 window:

```
#include <sys/types.h>
#include <X/Xhp.h>

int fildes;
char w_name[80];
XhpArgItem  arglist[] =
      {
      XhpENDLIST, 0
      };

XhpICreate(w_name, arglist);

fildes = gopen(w_name, OUTDEV, "Xn",INIT);
```

where the variable names are described as:

w_name	the path name to a regular file associated with the X10 window to be used by Starbase.
arglist	This array of type/value pairs describes the attributes of the window to be created. See the manual pages for `libXp(3X)` for further details in using these window utilities.

Device Dependent Characteristics

For device coordinate operations, location (0,0) is the upper left corner of the window at the time the window is `gopen()'d`. 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 raw device on which the window is located. These defaults are determined by the X10 server for that raw device. For example, an X10 server on a monochrome display defaults to only having one plane of color, black and white, while an X10 server on a color display utilizes as many planes of color as the display allows.

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

Line Type Defaults

Default line types are shown in the table below:

Table 1. - Default Line Types

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

Color Plane Defaults

All color planes for that physical device are enabled. You cannot disable planes through this driver.

Color Map Defaults

When an X10 window is gopen'd, the X10 colormap is read into the Starbase color map. A call to `inquire_color_table` provides information about the color map. The Starbase/X10 driver does not dynamically allocate colors in the X10 colormap. The X10 server, and direct calls to the X10 library, arbitrate over the allocation of color cells. It is recommended that the utility `xinitcolormap(1)` be used to initialize color cells in the X10 color map to an initial set of defined colors. Refer to `xinitcolormap(1)` manual page for further details. Note that `xinitcolormap` should be called as part of your X10 startup script, `.xstart`.

Input Defaults

The keyboard input is by default set to "cooked" mode. This mode returns ascii 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.

Regardless of the relative positions of the input devices on the HP-HIL loop, the X10 mouse is CHOICE and LOCATOR device ordinal one, a mask of all buttons pressed is

CHOICE and LOCATOR device ordinal two, and the X10 keyboard is CHOICE device ordinal three.

Starbase Functionality

Commands Not Supported (NO-OPS)

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

- await_retrace
- backface_control
- bank_switch
- bf_control
- bf_fill_color
- bf_interior_style
- bf_perimeter_color
- bf_perimeter_type
- bf_perimeter_repeat_length
- bf_surface_coeficients
- bf_surface_model
- dbuffer_switch
- def_color_table
- define_trimming_curve
- depth_cue
- depth_cue_color
- depth_cue_range
- display_enable
- double_buffer
- hidden_surface

- interior_style (INT_OUTLINE, INT_POINT)
- light_ambient
- light_attenuation
- light_model
- light_source
- light_switch
- shade_mode
- shade_range
- surface_coefficient
- surface_model
- viewpoint
- zbuffer_switch

Gescapes

The following gescapes are supported by this Starbase/X10 driver. The gescapes that are common to many display drivers, READ_COLOR_MAP, R_BIT_MASK, R_BIT_MODE, and R_DEF_FILL_PAT are documented in Appendix A. All other gescapes are used to control the input model. These input gescapes are described in the following section, Input Model.

- IGNORE_RELEASE
- READ_COLOR_MAP
- R_BIT_MASK
- R_BIT_MODE
- R_DEF_FILL_PAT
- TRIGGER_ON_RELEASE
- XN_INPUT_RAW
- XN_KEY_RELEASE
- XN_BUTTON_RELEASE

Input Model

A Starbase application is free to `gopen` an X10 window with the OUTDEV flag and utilize the X10 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 X10 input routines or Starbase input routines. Interchanging the two types of input calls in a single application may cause unexpected results.

The input model represents the X10 Window System as a virtual device including a keyboard and a three button mouse. The keyboard is accessed as a CHOICE device while the three button mouse is accessed as a CHOICE and a LOCATOR device.

The default mode returns ascii keycodes for all keystrokes. Special function keys return keycodes as defined by the Xrlib routine, `XrInputMap()`. These keycodes are detailed on the reference manual page `XrInputMap(3X)`. This is the "cooked" mode for input keystrokes. This input model largely obeys the same behavior as the `XrInputMap()` routine, thus it works correctly for most HP-HIL language keyboards. The only difference between the codes returned by this driver and `XrInputMap()` are that this driver returns a full set of ascii values. For example, the tab key generates its ascii value. This input model also performs as expected with non-HP X10 servers.

In "cooked" mode the mouse buttons are represented by values one through three, corresponding to the mouse button used. One represents the left hand mouse button, two represents the middle mouse button, and the right mouse button is represented by a three. In "raw" mode the value returned by a mouse button is determined by the "raw" value returned by the X10 server for that button.

The X10 Window System input device can also return raw keycodes. These codes are the unmapped keycodes returned by the X10 server. The X10 server model keymapping is defined in the file `/usr/lib/X/Xkeymap.defaul`. The input model can be placed in "raw" mode using the XN_INPUT_RAW gescape with an argument of TRUE. Similarly the input model can be reset to the default "cooked" mode using the XN_INPUT_RAW gescape with an argument of FALSE.

By default the Starbase/X10 driver returns events associated with the key presses and button presses. An application can request input events associated with both presses and releases of keys or buttons. Requesting the release events associated with mouse buttons or keys can be controlled independently.

An application can request events associated with both button presses and button releases by using the TRIGGER_ON_RELEASE or XN_BUTTON_RELEASE gescapes. The default mode of being sent only button press events can be reset using the XN_BUTTON_RELEASE gescape with an argument of FALSE or by using the IGNORE_RELEASE gescape. The value returned by a button release event is the

negative value of the corresponding button down event.

An application can request to be sent events associated with both key presses and key releases using the TRIGGER_ON_RELEASE or XN_KEY_RELEASE gescape. The default mode of being sent only the key press events can be reset using the XN_KEY_RELEASE gescape with an argument of FALSE or the IGNORE_RELEASE gescape. The value returned by a key release event is the negative value of the corresponding key down event.

Note that any HP-HIL pointing devices controlled by the X10 server are mapped into this virtual three button mouse X10 device. Note also that using the Xdevices description file allows a user to exclude a particular HP-HIL input device from use by the X10 server. This allows the application to then use a different Starbase device driver (e.g. HP-HIL) to access that excluded input device. Note, the mouse cannot be controlled by both the X10 server and the HP-HIL device driver, `libddhil.a`. If the mouse is to be controlled by the Starbase HP-HIL device driver then the mouse must be excluded from use by the X10 server.

One other key difference between most Starbase input device drivers and either the X10 library input routines or the input model of `libddXn.a` is that the latter do not work over a network. Most Starbase device drivers only work on a local machine. Thus it may not make sense to use the HP-HIL device driver if you also expect to use `libddXn.a` to access remote X10 servers for graphics displays.

Note: The input model has been changed from previous X10 releases to be compatible with the HP-HIL driver's behavior. For additional information on how the input portion of the driver behaves, see the documentation on the HP-HIL driver.

Programming Strategy

Directly calling the X10 library

It is acceptable for an application using the Starbase/X10 driver to make calls directly to the X10 library. In fact there are many instances where direct calls to X10 are preferred. In some cases calling an X10 library function is faster than calling a similar Starbase routine. For example, the X10 library call to `XFlush()` provides the exact same functionality as the Starbase call `make_picture_current()`. The call to `XFlush()` is more direct and faster. Similarly Starbase device coordinate output calls may be faster when made directly to the X10 library.

The only case where directly calling X10 routines may cause a problem occurs when an application is making X10 library input calls and also running the Starbase daemon to read

input. This is equivalent to the classical problem of having two processes reading input from a single device at the same time.

You are encouraged to make full use of the functionality of both the X window system and Starbase. You can use Starbase for 3D graphics or world coordinate systems, while using the X10 window system to define a cursor shape or to use an X10 font.

Window Resize

There are times when an application needs to know if the window in which it is running has changed size. There are two ways that an application can get this information.

First, if the application is running in the HP Windows-X environment, then all interesting window events are made available through the signal mechanism, SIGWINDOW. This mechanism emulates the previously available functionality of the HP Windows 9000 system.

Secondly, an application is free to handle the X10 input stream itself through calls in the X10 library, like `XNextEvent()`. If an application decides to handle the X10 input stream then it should `gopen` the X10 window using the kind argument OUTDEV. Each application should decide whether it wants to handle the X10 input stream itself, or whether it wants to execute Starbase input calls thus asking the Starbase/X library, `libddXn.a`, to handle the X10 events. If the application handles the X10 input stream itself then it is free to act on any X10 events it deems interesting, like window exposure or window focus change. If the Starbase/X10 library is handling the X10 events, then interesting window events, like window exposure, are not made available to the application.

Multiple Displays

It is possible for a single process to interact with windows on multiple displays. It is important to understand the notion used by the X10 library of currently active display. The X10 library provides a routine, `XSetDisplay()`, that changes the current display for a process. The Starbase/X10 driver never calls `XSetDisplay()`. Thus the burden of controlling the current display is left to the application.

It is necessary that a window be on the current display when it is `gopen()`'d. Thus a Starbase application controlling windows on multiple displays should set up a connection with a display, using `XSetDisplay()` or `XOpenDisplay()`, and then `gopen()` targeted windows on that display. If an application attempts to `gopen()` a window not on the current display, then either the `gopen()` will fail or some memory resources will be wasted.

Contents

X11

Using Starbase in X11 Windows

Chapter Uses

This chapter contains *device specific* information needed to run Starbase programs in X11 windows. If you need a general, non-device specific explanation of using Starbase in X11 windows, refer to the "Using Starbase with the X Window System" chapter of *Starbase Graphics Techniques.*

Do not confuse this chapter with SOX11 (the Starbase-on-X11 Device Driver). SOX11 is a driver that converts Starbase programs for use in X11 windows. Using the SOX11 translater may reduce the functionality and performance of your programs; nevertheless, they can be run over the network. This chapter describes how to run your Starbase programs in X11 windows with full functionality and performance, however, this cannot be done over the network.

Description

With the HP 9000 Series 300 6.5 HP-UX release and the HP 9000 Series 800 HP-UX 3.1 release, Starbase graphics was integrated with the X11 window environment. This environment supports a Starbase program running inside an X11 window with full Starbase functionality and performance comparable to raw mode (non-window) performance.

A key feature of Starbase programs operating in an X11 window is that any program can access any of the workstation input devices and use any of the display resources without interfering with other programs which are also accessing the same input and display resources. This ability to share the input and display resources is beneficial because it permits independently developed applications to run simultaneously in different X11 windows without interfering with each other.

Running Starbase in X11

When Starbase is rendered directly to an X11 window, the Starbase program and the X11 server must be run on the same machine. If remote Starbase is needed, use the Starbase-on-X Driver (described in the SOX11 chapter in this manual).

Changes Made with 3.1/6.5 Release

The process `/usr/lib/grmd` (the graphics resource manager daemon, which manages graphics resources used by Starbase, the X11 server, and HP Windows/9000) is started when the server is started or the display is opened in raw mode (except for the `hp9836a`, `hp98700`, and `hp98710` device drivers) or when the HP Windows/9000 window manager is started; it should be left undisturbed. When the last Starbase display driver is closed and the X server terminates, the `grmd` process terminates automatically.

Supported Visuals and Drivers

The following sections show the depths and visual classes supported by each display operating in the four server modes. The table shows which Starbase drivers can be used in the overlay and image planes for the different server operating modes.

Visuals Supported in the X11 Server Modes

There are four server modes: overlay mode, image mode, stacked screen mode, and combined mode. If a particular mode is not listed for a particular driver, the driver is not supported for that mode.

Refer to the *Starbase Graphics Techniques* chapter "Using Starbase with the X Window System" for more detailed information on visuals. Look in the section "Guidelines for Visuals" for information on what modes are simultaneously supported for different displays.

In the following descriptions, "n/n" indicates double buffering is supported, with "n" planes in each buffer. Note that backing store is not supported in all modes for Starbase.

HP 300 Hi-Resolution Displays

- Image Mode

  HP 318M — One plane with backing store for Xlib and Starbase. Supports `StaticGray`.

  HP 98544A — One plane with backing store for Xlib and Starbase. Supports `StaticGray`.

  HP 98545A — 2/2 planes with backing store for Xlib and Starbase. Supports `PseudoColor`.

  Four planes with backing store for Xlib and Starbase. Supports `PseudoColor`.

  HP 98547A — 3/3 planes with backing store for Xlib and Starbase. Supports `PseudoColor`.

  Six planes with backing store for Xlib and Starbase. Supports `PseudoColor`.

HP 300 Medium Resolution Display

- Image mode

  HP 98542 — One plane with backing store for Xlib and Starbase. Supports `StaticGray`.

  HP 98543 — 2/2 planes with backing store for Xlib and Starbase. Supports `PseudoColor`.

  Four planes with backing store for Xlib and Starbase. Supports `PseudoColor`.

HP 98548A Display

- Image mode

  One plane with backing store for Xlib and Starbase. Supports `GrayScale`.

HP 98549A and HP 319C Displays

- Image mode

3/3 planes with backing store for Xlib and Starbase. Supports `PseudoColor`.

Six planes with backing store for Xlib and Starbase. Supports `PseudoColor`.

HP 98550A Display

- Overlay mode

  Two overlay planes (three colors plus transparency) with backing store for Xlib and Starbase. Supports `StaticGray`.

- Image mode

  4/4 planes with backing store for Xlib and Starbase. Supports `PseudoColor`.

  Eight planes with backing store for Xlib and Starbase. Supports `PseudoColor`.

- Stacked Screen Mode

  Two overlay planes with backing store for Xlib and Starbase. Supports `StaticGray`.

  4/4 planes with backing store for Xlib and Starbase. Supports `PseudoColor`.

  Eight image planes with backing store for Xlib and Starbase. Supports `PseudoColor`.

HP 98720A Display

- Overlay mode

  Three overlay planes with backing store for Xlib and Starbase. Supports `PseudoColor`, eight colors.

- Image mode

  4/4 image planes with backing store for Xlib and Starbase. Supports `PseudoColor`.

Eight image planes with backing store for Xlib and Starbase. Supports `PseudoColor`.

8/8 image planes with backing store for Xlib only. Supports `PseudoColor`.

12/12 image planes with backing store for Xlib only. Supports `DirectColor`.

24 image planes with backing store for Xlib only. Supports `DirectColor`.

- Stacked screen mode

  Three overlay planes with backing store in Xlib and Starbase. Supports `PseudoColor`, eight colors.

  4/4 image planes with backing store in Xlib and Starbase. Supports `PseudoColor`.

  Eight image planes with backing store in Xlib and Starbase. Supports `PseudoColor`.

  8/8 image planes with backing store in Xlib only. Supports `PseudoColor`.

  12/12 image planes with backing store in Xlib only. Supports `DirectColor`.

  24 image planes with backing store in Xlib only. Supports `DirectColor`.

HP 98730A Display

- Overlay mode

  Three overlay planes with backing store in Xlib and Starbase. Supports `PseudoColor`, 8 colors.

  Four overlay planes with backing store in Xlib and Starbase. Supports `PseudoColor`, 16 colors.

- Image mode

  4/4 image planes with backing store in Xlib and Starbase. Supports `PseudoColor`.

  Eight image planes with backing store in Xlib and Starbase. Supports `PseudoColor`.

8/8 image planes with backing store in Xlib only. Supports `PseudoColor`.

12/12 image planes with backing store in Xlib only. Supports `DirectColor`.

24 image planes with backing store in Xlib only. Supports `DirectColor`.

- Stacked screen mode

  Three overlay planes with backing store in Xlib and Starbase. Supports `PseudoColor`, 8 colors.

  Four overlay planes with backing store in Xlib and Starbase. Supports `PseudoColor`, 16 colors.

  4/4 image planes with backing store in Xlib and Starbase. Supports `PseudoColor`.

  Eight image planes with backing store in Xlib and Starbase. Supports `PseudoColor`.

  8/8 image planes with backing store in Xlib only. Supports `PseudoColor`.

  12/12 image planes with backing store in Xlib only. Supports `DirectColor`.

  24 image planes with backing store in Xlib only. Supports `DirectColor`.

- Combined mode

  Three overlay planes with backing store in Xlib and Starbase. Supports `PseudoColor`, 7 colors.

  Four overlay planes with backing store in Xlib and Starbase. Supports `PseudoColor`, 15 colors.

  4/4 image planes with backing store in Xlib and Starbase. Supports `PseudoColor`.

  8 image planes with backing store in Xlib and Starbase. Supports `PseudoColor`.

  8/8 image planes with backing store in Xlib only. Supports `PseudoColor`.

  12/12 image planes with backing store in Xlib only. Supports `DirectColor`.

24 image planes with backing store in Xlib only. Supports `DirectColor`.

Drivers Supported in the X11 Server Modes

In the following table, if a particular driver is not listed for a particular display and server operating mode, the driver is not supported for that configuration.

Table X11-1. Supported Drivers in X11 Server Modes

Display	Supported Driver	Overlay Mode	Image Mode	Stacked Screen Mode	Combined Mode
HP 300 Hi-Res Displays	`hp300h`		yes		
HP 300 Medium Res Displays	`hp300l`		yes		
HP 98548A	`hp98550`		yes		
HP 98549 HP 319C	`hp98550`		yes		
HP 98550A	`hp98550` and `hp98556`	Image (raw mode only) and overlay planes	Image plane (no raw mode)	Overlay and image planes	

Table X11-1. Supported Drivers in X11 Server Modes Continued

Display	Supported Driver	Overlay Mode	Image Mode	Stacked Screen Mode	Combined Mode
HP 98720	hp98720	Image (raw mode only) and overlay planes	Image plane (no raw mode)	Overlay and image (no raw mode) planes	
	hp98721	Image plane (raw mode only)			
HP 98730	hp98730	Image (raw mode only) and overlay planes	Image plane (no raw mode)	Image plane (no raw mode)	Image and overlay planes (no raw mode)
	hp98731	Image planes (raw mode only)	Image plane (no raw mode)		Image plane (no raw mode)

Note

In image mode, raw mode access to the overlay planes is not supported. Graphics in the overlay planes may obscure the window system and may interfere with overlay-plane cursor operation.

The HP 98720w driver, developed to support HP Windows/9000, cannot be used with any server mode.

X11 Cursors and Starbase Echoes

The Starbase `gescape R_OVERLAY_ECHO` can be used to change the default echo placement from the image planes to the overlay planes. The term "shares", used for some drivers below, means the echo or cursor is rendered into the same planes used by Starbase or Xlib for rendering.

The following list shows default positions where the Starbase echo and X11 cursor (called echo and cursor, respectively) reside for each of the X11 server operating modes.

HP 300 Medium and High Resolution Displays

- Image Mode

  Echo and cursor share the image planes.

HP 98548A and HP 98549A Displays

- Image Mode

  Echo and cursor share the image planes.

HP 98550A Display

- Overlay Mode

  Echo placed in opened image or overlay planes. Cursor resides in overlay planes.

- Image Mode

  Echo and cursor share the image planes.

- Stacked Screen Mode

  If overlay-plane window is opened, echo and cursor share overlay planes. If image-plane window is opened, echo and cursor share image planes.

HP 98720 Display

- Overlay Mode

  If X11 window overlay-plane is opened, echo shares three overlay planes.

  If image planes are opened, raster echo resides in image plane.

  If image planes are opened in raw mode, vector echo resides in cursor plane.

  Cursor shares three overlay planes.

- Image Mode

  If X11 window image-plane is opened:
  - raster echo resides in image planes.
  - vector echo resides in cursor planes.

  Cursor shares image planes.

- Stacked Screen Mode

  If overlay plane in X11 windows is opened, echo shares three overlay planes.

  If image plane in X11 windows is opened:
  - raster echo resides in image planes.
  - vector cursor resides in cursor plane.

  Cursor:
  - shares image plane for image-plane window.
  - shares overlay plane for overlay-plane window.

HP 98730 Display

- Overlay Mode

  If overlay-plane X11 window is opened, echo shares three or four overlay planes.

  If image planes are opened and X11 uses three overlay planes, vector echo resides in cursor plane.

  If image planes are opened and X11 uses four overlay planes, vector echo resides in image planes.

  X11 cursor uses hardware cursor.

- Image Mode

  If image-plane X11 window is opened, raster echo resides in image planes and vector echo resides in cursor plane.

  X11 cursor uses hardware cursor.

- Stacked Screen Mode

If image-plane X11 window is opened, echo shares three or four overlay planes.

If image-plane X11 window is opened, raster echo resides in image planes.

If X11 uses three overlay planes and image planes are opened, vector echo resides in cursor plane.

If X11 uses four overlay planes and image planes opened, vector echo resides in image planes.

X11 cursor uses hardware cursor.

- Combined Mode

  If overlay-plane X11 window is opened, echo shares three or four overlay planes.

  If image-plane X11 window is opened, raster echo resides in image planes.

  If image-plane X11 window is opened and X11 uses three overlay planes, vector echo resides in cursor plane.

  If image-plane X11 window is opened and X11 uses four overlay planes, vector echo resides in overlay planes.

  X11 cursor uses hardware cursor.

HP 98731 Display

The `hp98731` driver cannot open an X11 overlay-plane window.

- Overlay Mode

  If image planes are opened and X11 uses three overlay planes, echo resides in cursor plane.

  If image planes are opened and X11 uses four overlay planes, echo is not supported.

  X11 cursor uses hardware cursor.

- Image Mode

  If image-plane X11 window is opened, echo resides in cursor plane.

X11 cursor uses hardware cursor.

- Stacked Screen Mode

  If image planes are opened and X11 uses three overlay planes, echo resides in cursor plane.

  If image planes are opened and X11 uses four overlay planes, echo is not supported.

  X11 cursor uses hardware cursor.

- Combined Mode

  If image-plane X11 window is opened and X11 uses three overlay planes, echo resides in cursor plane.

  If image-plane X11 window is opened and X11 uses four overlay planes:

  - vector echo resides in overlay planes.
  - raster echo not supported.

  X11 cursor uses hardware cursor.

Shared Memory Usage

Note The "Shared Memory Usage" section applies to Series 300 computers only.

Display frame buffers and Starbase shared memory resources use a range of addresses on Series 300 computers. Both the virtual address space and system shared memory are limited resources that can affect how complicated window and graphics environments operate. This section describes how to control these resources so your program will operate more efficiently.

A typical user graphics process on Series 300 has a virtual memory address space as shown in the following figure.

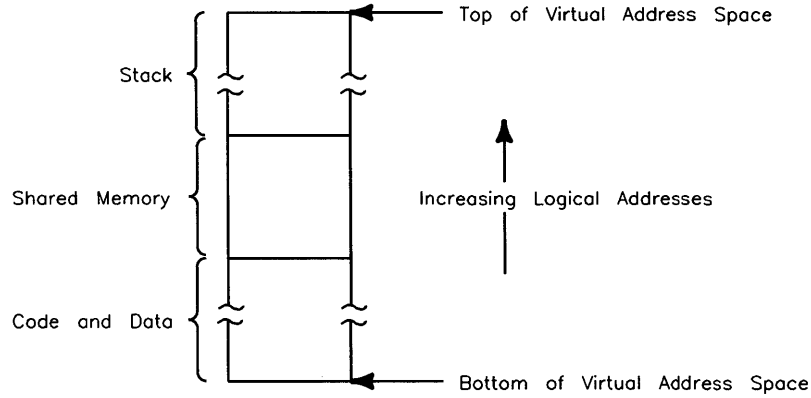


Figure X11–1. Virtual memory map in a graphics process

Stack space starts near the top of virtual memory, and the stack grows down from the top. Code and data space, including the dynamically allocated data space, or "heap", start at the bottom (address 0) of virtual memory and grow up from the bottom. Located between these two areas is graphics and application **shared memory.** The "Memory Management" section, Chapter 3, in the *HP-UX System Administrator Manual* describes the organization of shared memory in detail. This discussion assumes no space is required for shared memory allocated by the application. Developers and users of programs that need additional shared memory must account for it in the address and space computations they perform.

Window Processes

Before discussing why and how graphics shared memory is used, you must understand what a **graphics process** is. In general, a graphics process is any process that meets one of the following conditions:

1. The process generates Starbase graphics to a frame buffer device or a window.
2. The process is the X11 server.

How Do Graphics Processes Use Shared Memory?

Graphics processes use shared memory to access data pertaining to the display device and the X11 resources created by the server (resources include windows, color maps, and cursors). In addition, Starbase uses shared memory to communicate between the main process and the asynchronous input/tracking daemon. The physical graphics display console space and frame buffer are mapped as shared memory so that multiple processes can share them.

The graphics process (X11 server or Starbase application) that starts up first on the system initiates an independent process called the Graphics Resource Manager (GRM). The GRM helps the server and Starbase process manage graphics shared memory. The GRM continues to run as long as any graphics process is running, but terminates automatically (returning any allocated system resources) when all graphics processes end.

Shared Memory Problems

Two common problems encountered with graphics shared memory are:

Shared memory is too small

Code and data space is too small (shared memory is positioned too low in the virtual address space)

Shared Memory Size

Usually the contiguous block of shared memory used by the GRM is two megabytes (2Mb) in size. One problem encountered with shared memory is that it just isn't large enough for some applications.

When an application attempts to use more shared memory than is available (e.g., for retained rasters for newly created graphics windows), the shared memory "get" fails and the application terminates. This does *not* affect the window system, except that you run into a limit for the number of retained-raster graphics windows you can create.

For example, suppose you have an application that uses many graphics windows with retained rasters. Retained rasters are costly in terms of memory usage. Each pixel of the raster consumes a byte of memory (except on monochrome displays). Therefore, a window that is 1024×512 pixels consumes a half-megabyte (0.5Mb)

of memory. At this rate, graphics shared memory will be completely consumed by four graphics windows (4 windows × 0.5Mb = 2.0Mb). Also, remember to take into account the shared memory needed for other communication with the X11 server.

You can circumvent this problem via certain environment variables which control the size and location of shared memory, as discussed in the remaining sections of this chapter.

Code and Data Space

Your actual graphics program resides in the contiguous block of memory immediately below shared memory—the code and data space. By default the code/data space is 8.75Mb in size.

Each graphics process must have a code/data space that will fit into the area below shared memory. In other words, the shared memory must start at some address such that the maximum code/data space of the process will not interfere with the shared memory space.

For example, suppose you've written a C program to perform graphics in a window. The program does many program-controlled heap manipulations, so it must allocate (via *malloc(3)*) enormous amounts of dynamic data space. The program's code is 2Mb in length, but as it runs, it consumes up to 8Mb of dynamic data space (heap)—a total of 10Mb for both the code and its data. This will exceed the default maximum code/data space size by approximately 1.25Mb (10Mb − 8.75Mb max size = 1.25Mb over max).

This problem can be surmounted via the `SB_DISPLAY_ADDR` and `WMSHMSPC` environment variables (both discussed in a later section). By moving shared memory upward, you create more room for the code/data space.

A Close-Up of Shared Memory

Before discussing how to change shared memory to circumvent shared memory problems, you should understand how the shared memory is organized. The following figure illustrates its structure.

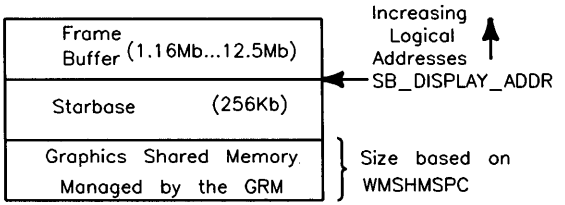


Figure X11-2. Graphics Shared Memory

The following table briefly describes the three components of shared memory.

Table X11-2. Windows Shared Memory Close-Up.

Component	Description
Frame Buffer	Corresponds to the control registers and screen of your display. This area may range from a little over 1Mb to as much as 12.5Mb for an HP 98730 display.[1]
Starbase	Used by the Starbase Graphics system (uses include the event queue, and communication between the tracking daemon and the main program).
GRM Shared Memory	Stores information pertinent to X11 windows and shared use of the display (for example, retained rasters for graphics windows, cursor information, and fonts).

[1] If more than one display is accessed at the same time (for example, an X11 server with two "heads", or a Starbase program that gopens multiple displays), you must add the space required for each.

Shared Memory Environment Variables

Two environment variables—SB_DISPLAY_ADDR and WMSHMSPC—control the location and size of graphics shared memory. To configure the shared memory, you must change the value of the appropriate variable(s) *before* starting (or restarting) the graphics application or X11 server.

SB_DISPLAY_ADDR Variable

SB_DISPLAY_ADDR points to the address immediately above the Starbase area of shared memory. You can change the position of the shared region by changing the value of SB_DISPLAY_ADDR, since the shared region is positioned relative to this address. If you do not set the value of SB_DISPLAY_ADDR, it defaults to 0xB00000 (11Mb), which leaves approximately 8.75Mb for code/data space.

WMSHMSPC Variable

The other environment variable for configuring shared memory is WMSHMSPC, which determines the size of the GRM-managed area of the shared region.

It is important to know that the WMSHMSPC variable only has an effect on allocated shared memory in the environment in which the GRM is started. For example, if no graphics process is running and you start up the X11 server, the value of WMSHMSPC in that environment will be used in allocating the shared memory for the GRM. Other graphics processes that will run simultaneously should be given the same value of WMSHMSPC, however, so that they can correctly compute the virtual address to attach the GRM shared memory.

Changing Shared Memory

If the default configuration for shared memory, as defined by the environment variables SB_DISPLAY_ADDR and WMSHMSPC, is inadequate for your system, you should set these variables accordingly. In general, you should use the following rules when reconfiguring windows shared memory:

- If you have processes requiring more shared window space than the default 2Mb, increase the value of WMSHMSPC to accommodate the amount of shared memory required.

- If you have processes that require more code and data space than the default 8.75Mb, increase the value of SB_DISPLAY_ADDR so the processes will fit in the code/data space.

Consequences for Changing Variables

Because the WMSHMSPC and SB_DISPLAY_ADDR environment variables are closely related, changing one may affect the other. Before changing a variable, you should understand the possible side effects.

"WMSHMSPC"

If you increase the value of WMSHMSPC but do *not* change the value of SB_DISPLAY_ADDR, you effectively decrease the size of the code/data space by the increased size of the shared area because the GRM shared memory area (defined by WMSHMSPC) resides below SB_DISPLAY_ADDR. Therefore, increasing the size of this area while holding SB_DISPLAY_ADDR constant shrinks the code/data space, as shown in the following figure.

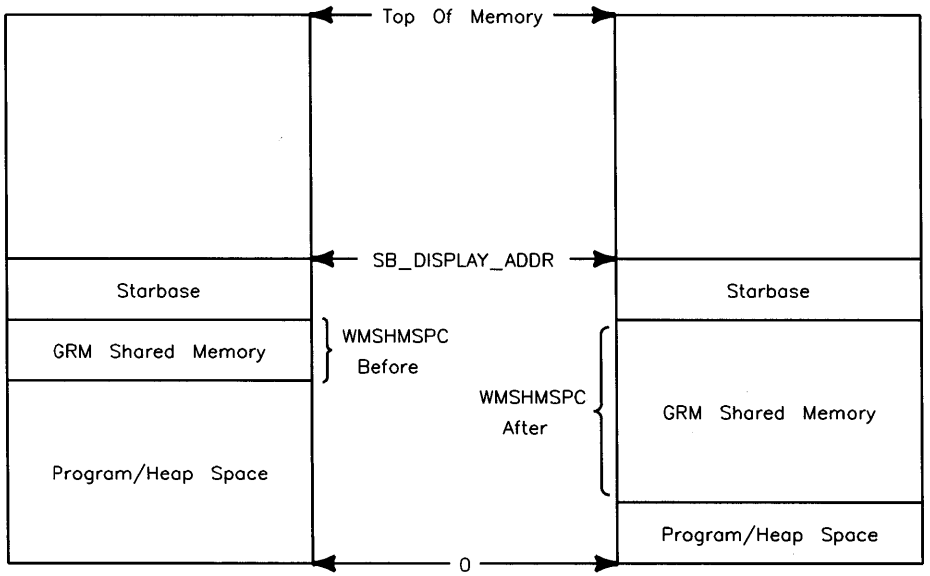


Figure X11-3. Squeezing Code/Data Space via "WMSHMSPC"

Conversely, if you decrease the size of shared memory while holding the SB_DISPLAY_ADDR variable at a constant value, the code/data space will increase by the change in WMSHMSPC. Certain operations may fail if you set WMSHMSPC to a value less that 0.5Mb. The minimum required value of WMSHMSPC increases as you increase the number of X11 resources and Starbase processes.

The following equation shows the relationship between code/data size and the variables WMSHMSPC and SB_DISPLAY_ADDR.

```
code/data space = SB_DISPLAY_ADDR - 256Kb (for Starbase) - WMSHMSPC
```

You must change SB_DISPLAY_ADDR if you do not want the code/data space to change when you change the value of WMSHMSPC. For example, if you increase WMSHMSPC from the default 2Mb to 3Mb, you must also increase SB_DISPLAY_ADDR from the default 11Mb to 12Mb.

Kernel Configuration Limitations

Even though Starbase and the X11 server allow you to move and change the size of shared memory, there are still some limitations to its location and size. These limitations are defined by kernel configuration parameters. Consult the *HP-UX System Administrator Manual* for details on setting and changing these parameters.

shmmaxaddr Variable

The maximum (highest) address allowable for any shared memory is defined by the shmmaxaddr kernel configuration variable. By default shmmaxaddr is defined as 0x1000000 (16Mb). This means the topmost address of all of graphics and application shared memory cannot equal or exceed the value of shmmaxaddr, as shown in the following figure.

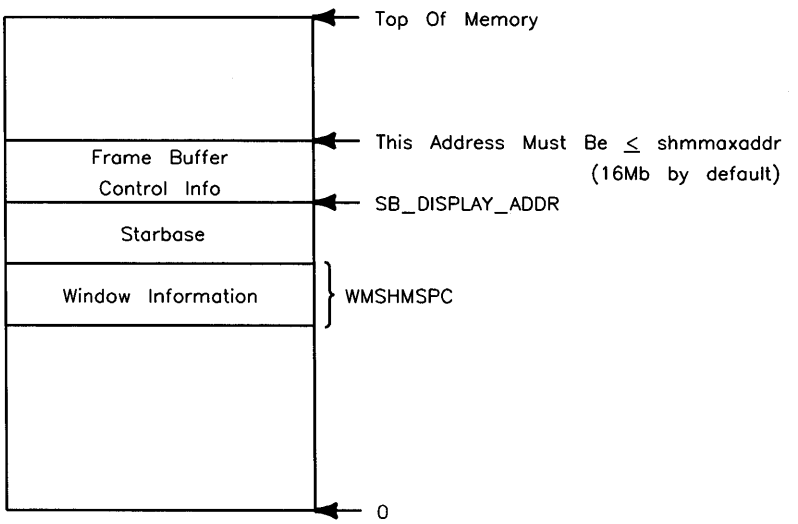


Figure X11-4. The "shmmaxaddr" Variable and Shared Memory

The following equation gives the relationship between `shmmaxaddr` and the `SB_DISPLAY_ADDR` variable:

```
SB_DISPLAY_ADDR < shmmaxaddr - stack space under shmmaxaddr
        - frame buffer and control space size
```

shmmax Variable

The `shmmax` kernel configuration variable defines the maximum allowable size of any shared memory segment. By default, this value is set to `0x600000` (6Mb). This does *not* mean shared memory is limited to 6Mb maximum; it merely means each *segment* is limited to 6Mb.

The `shmmax` variable must be set to the largest of any segment size your program will allocate. It must be at least as large as `WMSHMSPC`.

Example for Using Variables

The following example should assist you in understanding how to use these variables. Suppose you've written two C-language programs to perform Starbase graphics with graphics windows. Program one (`prog1.c`) will require large amounts of code/data space when it executes—greater than the default amount. Program two (`prog2.c`) won't use as much code/data space as `prog1.c`, but

makes heavy use of retained-raster graphics windows, and requires more GRM area than the default. Detailed descriptions of each program follow.

Program One ("prog1.c")

When compiled, `prog1.c`'s executable code size is just under 2Mb. For simplicity, round the code size to 2Mb (`0x400000`).

When the program executes, it allocates large amounts of dynamic data space (via *malloc(3)*). Through diligent calculations you've determined that the program could theoretically allocate up to 6.5Mb of memory from the dynamic data space (heap). Again, for simplicity, round this figure to 7Mb (`0x700000`).

The maximum code/data space consumed by this application is 9Mb, computed as follows:

```
  0x200000      (2Mb for executable code)
+0x700000       (7Mb for the program's data)
========
  0x900000      (9Mb total program and data space)
```

Program Two ("prog2.c")

When running, `prog2.c` will create a maximum of four graphics windows, each with a retained raster with dimensions 1024 by 512 pixels. The retained raster for each window consumes 0.5Mb; therefore the maximum amount of memory required by the retained rasters alone is 2Mb (4 windows at 0.5Mb each = 2Mb).

When determining the size of the window information area, you should also consider that other windows will use the area also. For example, fonts for all windows are loaded into the GRM area. To be safe, add 1Mb to the size of the GRM area to compensate for other needs.

For this example, it gives a maximum size of 3Mb for the GRM shared memory area—1Mb larger than the default value for `WMSHMSPC`.

Determining the Correct Value for "SB_DISPLAY_ADDR"

Now you know the maximum code/data space size (9Mb) and the maximum size of the window information area of shared memory (`WMSHMSPC` = 3MB), so you can determine the correct value for `SB_DISPLAY_ADDR`.

SB_DISPLAY_ADDR should be computed as follows:

```
SB_DISPLAY_ADDR = code/data size + WMSHMSPC
     + 256Kb (for the Starbase area)
```

Using this equation, SB_DISPLAY_ADDR is computed to be 14.25Mb:

```
  0x900000     (9Mb code/data space for prog1.c)
+0x300000      (3Mb window information area—WMSHMSPC)
+0x040000      (256Kb for the Starbase area)
========
  0xc40000     (12.25Mb = SB_DISPLAY_ADDR)
```

Ensuring Correct Values

Before actually setting these values, you should be sure the values will not cause detrimental side effects; mainly, shared memory must be within the **shmmaxaddr** value.

Using equation 3, ensure that the window shared memory area is within the bounds set by the **shmmaxaddr** kernel configuration variable:

```
 0x1000000     (default value for shmmaxaddr)
−0x220000      (maximum frame buffer size)
=========
 0xdd0000      ≥ SB_DISPLAY_ADDR (0xc40000)
```

SB_DISPLAY_ADDR also passes on this test: the window system shared memory is located below **shmmaxaddr**.

Note that the actual mapped size of the frame buffer varies from display to display. For DIO-I displays, the maximum is 0x220000. For DIO-II displays, the maximum is 0xc00000. For more information on DIO-I and DIO-II displays, refer to your display's chapter in this manual.

Increasing Performance by Decreasing Memory

The previous discussion has centered mainly on increasing memory size to ensure that all graphics processes can execute without running into shared memory problems. At the other end of the spectrum, you may be able to reduce virtual memory usage, thus increasing system performance.

Decreasing memory requirements is practical when most of your graphics processes are small and when your window shared memory requirements are minimal. For example, if none of your processes use more than 4Mb of code/data

space (4.75Mb less than the default amount available), and you don't need shared memory for retained rasters, you can set SB_DISPLAY_ADDR and WMSHMSPC to values less than their defaults, thus increasing the performance of your applications.

IMPORTANT When computing the amount of code/data space required, keep in mind that the X11 server is also a graphics process in the same process group as other graphics processes. Therefore, you must leave enough code/data space for the server to execute.

Device Specific Characteristics

Monochrome Color Map Changes

When using the hp300h and hp300l device drivers in an X11 window on a monochrome display, the define_color_table routine only changes the internal Starbase color map, and does not affect the X11 color map.

Gescapes

Using the R_LOCK_DEVICE, R_UNLOCK_DEVICE, and SWITCH_SEMAPHORE gescapes is not recommended when using an X11 window. The result of using them is that the Starbase program and the X11 window system may get into a deadlocked state. You can recover from this state by killing the Starbase program.

If you need to use these gescapes, do not call make_picture_current while the user program has the device locked, i.e., when an R_LOCK_DEVICE gescape has been called but no matching R_UNLOCK_DEVICE has been called. The procedure flush_buffer is an alternative to make_picture_current for some devices.

Moving into X11

Moving from X10 to X11

Starbase in an X10 window is only supported by the Xn driver, which converts Starbase calls to X10 protocols. In moving a Starbase program to X11, you have two choices:

1. Re-link with the SOX11 driver. This will support remote Starbase.
2. Use Starbase directly in an X11 window. You can link your Starbase program with the 3.1/6.5 (or later) software release Starbase drivers so that the program operates directly in an X11 window. This will greatly increase performance, but will only work when the Starbase program and server run on the same system. You must still use the SOX11 driver to render Starbase graphics to a remove X server.

Moving from X11 Revision A.00 to X11

Starbase in an X11 revision A.00 window is supported by both the Xn driver and the SOX11 driver. In moving the Starbase program to X11, you have two choices:

1. Continue using the SOX11 driver. If your approach requires remote Starbase you will want to continue with this approach. You must use the SOX11 drivers if you only have access to the executable and cannot link in other drivers.
2. Use Starbase directly in an X11 window. You can link your Starbase program with the 3.1/6.5 (or later) software release Starbase drivers so that the program operates directly in an X11 window. This will greatly increase performance, but will only work when the Starbase program and server run on the same system. You must still use the SOX11 driver to render Starbase graphics to a remote X server.

X11 Documentation

The following references are helpful when working with X11 windows:

- *A Beginner's Guide to the X Window System*
- *Configuring the X Window System*
- *Programming with the HP XWidgets and XIntrinsics*
- *Programming with Xlib, Version 11*
- *Programming with Xrlib*
- *Starbase Graphics Techniques*
- *Starbase Programming with X11*
- *Using the X Window System, Version 11*
- *X11 Programming Manual* by O'Reilly
- *X11 Reference Manual* by O'Reilly
- *X Window System User's Guide* by O'Reilly
- *Xlib Quick Reference Guide*

Win an HP Calculator!

Your comments and suggestions help us determine how well we meet your needs. **Returning this card with your name and address enters you into a quarterly drawing for an HP calculator***.

Starbase Device Drivers
Library Manual

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

You have used this product:

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

fold

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

Comments: _____

*Offer expires June 1990. (98592-90018 E0989)

Please Tape Here

Please print or type your name and address.

Name: _____

Company: _____

Address: _____

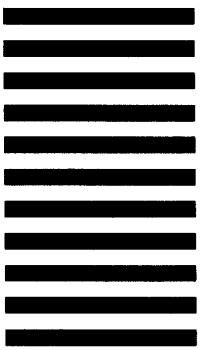
City, State, Zip: _____

Telephone: _____

Additional Comments: _____

Starbase Device Drivers Library Manual
HP Part Number 98592-90018
E0989

NO POSTAGE
NECESSARY
IF MAILED
IN THE
UNITED STATES



BUSINESS REPLY MAIL

FIRST CLASS PERMIT NO. 37 LOVELAND, COLORADO

POSTAGE WILL BE PAID BY ADDRESSEE

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

HP Part Number
98592-90018
Microfiche No. 98592-99018
Printed in U.S.A. E0989

For Internal Use Only