Starbase Programming with X11

HP 9000 Series 300/800 Computers

HP Part Number 98592-90000

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.

December 1988 ... Edition 1. This manual is valid for HP-UX release 6.5 on all HP 9000 Series 300 Models, and HP-UX release 3.1 on all HP 9000 Series 800 Models.

Preface

With the HP 9000 Series 300 6.5 HP-UX release and the HP 9000 Series 800 HP-UX 3.1 release, Starbase graphics has been integrated with the X11 window environment. This environment supports a Starbase program running inside of an X11 window with full Starbase functionality and performance comparable to raw mode (non-window) performance. This manual is intended to help you develop Starbase programs that run in an X11 window. Information is also provided to enable you to move graphics applications from other window environments (for example, X10) to the X11 environment.

This manual provides the following:

- A description of the **window systems** supported on Hewlett-Packard's HP-UX workstations and the graphics libraries that are supported in each window system.
- A description of the following topics, with the focus being on Starbase graphics in an X11 window:
  - Window system architecture
  - Graphics output
  - Raster text operation
  - Input operation
  - Graphics hardcopy operation
- Application development guidelines to assist you in developing Starbase programs that run in an X11 window.

Note that programming examples in this manual are used primarily to demonstrate new programmatic capabilities. However, where the programmatic interface has not changed but the functionality has changed, the manual focuses on describing the functionality changes. For example, the Starbase double-buffer procedures and parameters have not changed; however, the functionality provided by these procedures in an X11 window differs slightly from the functionality provided in raw mode. Therefore, this manual describes the functionality differences between raw mode operation and operation in an X11 window.

Audiences

While this manual is aimed primarily at users who already know Starbase and X11, some introductory material is provided so that programmers who are new

to Hewlett-Packard's HP-UX workstation will be able to obtain an overview of the supported graphics libraries and window systems.

Releases of X11

Starbase graphics was not fully integrated into the initial release of X11. This initial release of X11 is designated in this manual as "X11 revision A.00.*dd*" The "00" indicates that this is the first release. The "*dd*" denotes two decimal digits used internally by Hewlett-Packard; these can be ignored. The release of X11 which supports Starbase graphics in an X11 window is designated as "X11 revision A.01.*dd*".

When X11 is referenced in this manual, it refers to the release which supports X11 revision A.01.*dd*. When the initial release is discussed, it is explicitly referred to as "X11 revision A.00". Because X11 was introduced with the 3.1 (Series 800) and 6.5 (Series 300) HP-UX releases, "3.1/6.5" is used to refer to these releases. "Pre-3.1/6.5" is used in discussing capabilities prior to these releases.

If you're not sure which version of X11 you have, you can use the `what` command, as follows:

```
what  /usr/bin/X11/X
```

This returns the revision number of the X11 server. It returns a revision number of the form `A.01.`*dd* if you are using a tape, and a number of the form `A.01` otherwise.

Recommendations

It is recommended that new Starbase applications be designed to run in an X11 window. In developing a Starbase program to run in an X11 window, you should be familiar with both how Starbase works in raw (non-window) mode and how X11 works. This manual should then be referenced for information on how Starbase works in an X11 window.

Application Sharing of Workstation Resources

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.

Contents

1

Graphics Libraries and Window Systems

This chapter describes the graphics libraries and window systems that are supported on the HP-UX workstations. For information on which displays are supported, refer to the *Starbase Device Drivers Library Manual*

The graphics libraries supported on the HP 9000 Series 300 and Series 800 workstations are:

- Starbase Graphics Library
- Starbase Display List
- HP-GKS
- AGP/DGL
- Xlib Graphics

The window systems which are supported on the HP 9000 Series 300 and/or the Series 800 workstations are:

- HP Windows/9000 (Series 300 only)
- X10
- X11 revisions
    - X11 revision A.00
    - X11 revision A.01

These window systems are described in detail in the "Window Systems" section of this chapter.

Graphics Libraries

The different graphics libraries which are supported on Hewlett-Packard's HP-UX workstations are:

- **Starbase:** Starbase is Hewlett-Packard's 2D and 3D graphics library. It provides a wide variety of input and output capabilities and offers very high performance. Starbase graphics operations are based on three coordinate systems: device (display) coordinates, floating point virtual coordinates and integer virtual coordinates. Advanced 3D capabilities include hidden-surface removal, shading, and light sources. Starbase is supported by a wide variety of device drivers that permit your Starbase program to output to graphics terminals, plotters, and bitmapped displays. The Starbase drivers are discussed in the *Starbase Device Drivers Library Manual*
- **Starbase Display List**Starbase Display List:| The Starbase display list is built "on top of" Starbase and is used to organize, manipulate and execute Starbase graphics operations. For example, an interactive graphics editor could take the user's inputs from a data tablet and use Display List to store Starbase procedures in a data structure which corresponds to these inputs. Subsequently, when the user wants to view the output, the Display List can be traversed to generate the image on a display.
- **HP-GKS:** HP-GKS is a 2D graphics library. It is Hewlett-Packard's implementation of the ANSI Graphical Kernel System.
- **AGP/DGL:** DGL (Device-Independent Graphics Library) is a 2D graphics library. AGP (Advanced Graphics Package) is a 3D graphics library. AGP resides "on top of" DGL (that is, AGP calls are translated into DGL calls). DGL and AGP are typically referenced together as "AGP/DGL" because of this architectural connection. AGP and DGL are only recommended when you are porting an existing application; Starbase is recommended for new applications.
- **Xlib Graphics:** The X Window System library (called "Xlib") provides a 2D device-coordinate graphics interface. Xlib provides vector-, raster- and input operations within an X window.

Building on Starbase

Starbase forms the platform for DGL, HP-GKS and the Starbase Display List; i.e., DGL, HP-GKS and the Starbase Display List are supported in Starbase. DGL and HP-GKS calls are translated into Starbase calls. Because Starbase is, in turn, built on top of many different device drivers, HP-GKS and DGL can access most of the devices that Starbase supports. Refer to HP-GKS and DGL documentation for a list of Starbase devices which are accessible by HP-GKS and DGL.

Because AGP resides on top of DGL, AGP is also built (indirectly) on the Starbase platform (separated by the DGL layer). The following diagram shows this architecture:

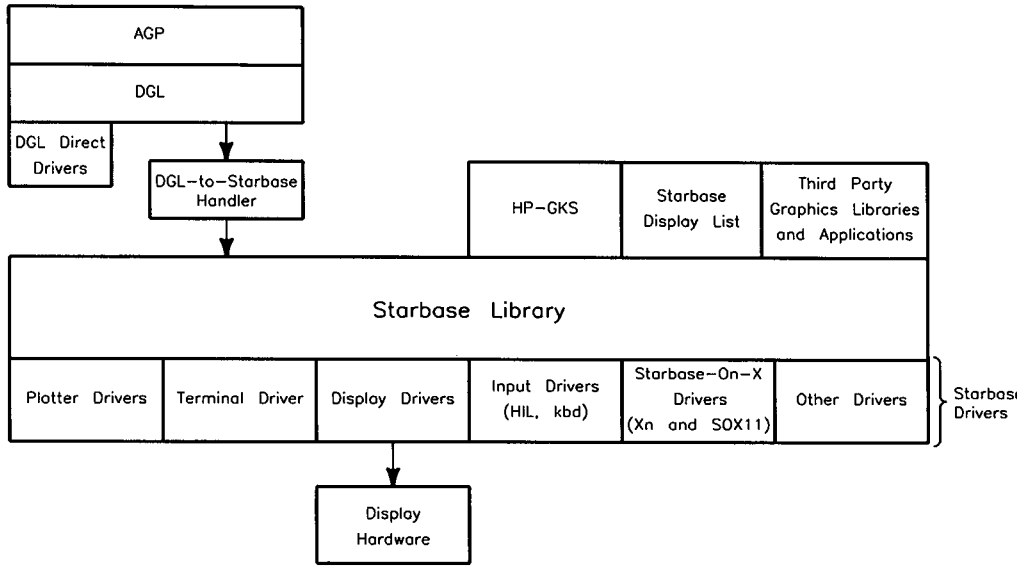


Figure 1-1. Graphics Libraries Built on Top of Starbase

Graphics Libraries Notes

1. The Display Drivers communicate directly to the bitmapped display hardware. Refer to the *Starbase Device Drivers Library Manual* for a discussion of the supported display drivers.

2. DGL, and indirectly, AGP, also support direct drivers. That is, there are drivers which permit DGL programs to communicate directly with the graphics hardware without using Starbase and the Starbase drivers.
3. Many third-party graphics libraries and application programs are built on top of the Starbase platform. For information on these libraries and application programs, see your Hewlett-Packard sales representative.
4. The Starbase-On-X driver translates Starbase calls into Xlib calls. For example, the Starbase call `draw2d` is translated by this driver into the Xlib call `XDrawLine`. This driver permits Starbase to work in an X window. However, because not all Starbase calls (for example, 3D solids-modeling calls) can be translated into Xlib calls, this driver does not support full Starbase functionality. However, since Xlib works over the network between a client and a server, the Starbase-On-X driver does permit Starbase to work over the network, albeit with reduced functionality and performance compared to Starbase running with the Starbase display drivers. There are two versions of the Starbase-On-X Driver:
    - *Xn Driver:* This Starbase driver translates Starbase calls into X10 Xlib calls and was introduced with X10.
    - *Starbase-on-X11 (SOX11) Driver* This Starbase driver translates Starbase calls into X11 Xlib calls and was introduced with X11 revision A.00.

For more details on the Xn and SOX11 drivers, refer to the *Starbase Device Drivers Library Manual.*

Window Systems

The window systems supported on HP-UX workstations are only supported on bitmapped displays, not on graphics terminals. In the case of **raw-mode** operation, which means no window system at all is running, a graphics application has control of the entire display.

The window systems plus raw-mode operation are described below:

- **Raw Mode:** In raw mode, the graphics application has control of the entire display. When moving applications from raw mode to the X11 environment, application developers need to know how graphics operation in an X11 window differs from graphics operation in raw mode. A large part of this manual is devoted to this subject.
- **HP Windows/9000:** This window system was developed by Hewlett-Packard for the HP 9000 Series 300 computers and is not supported on the HP 9000 Series 800 computers. HP Windows/9000 supports terminal emulator windows and graphics windows on a local workstation only. This means that the program making window calls must be on the same computer that is running the window system.
- **X10:** X10 was developed at MIT and was introduced by Hewlett-Packard in March, 1987. X10 provides a distributed (network-transparent) window system. The program making Xlib calls can be on a different computer than the program running the window system.
- **X11 revisions:**
  - **X11 revision A.00:** X11 was also developed at MIT. X11 release A.00 was introduced by Hewlett-Packard in June, 1988 and was initially supported with the 6.2/3.0 release.
  - **X11 revision A.01:** This release of X11 supports full-functionality Starbase in an X11 window with performance comparable to raw-mode Starbase. Starbase graphics became fully integrated with X11, but from a programmer's point of view, the two X11s are the same. X11 release A.01 is the version discussed in this manual.

The term **X Window System** is a capability that is common to X10, and to both revisions of X11.

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 1-1. 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

2

Graphics and Window System Architecture

The architectural information in this chapter is provided to assist you in understanding the different local and remote graphics, and the windows capabilities provided on the HP-UX workstation, and to assist you in moving applications to the X11 environment.

Raw Mode Starbase Architecture

The architecture of raw mode Starbase is shown in the following diagram:

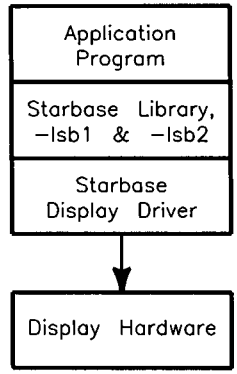


Figure 2-1. Raw Mode Starbase Architecture

Notes on Raw Mode Starbase Architecture

1. The application program, Starbase library, and display driver execute as one process.
2. This architecture permits Starbase to write directly to the display hardware via the Starbase display driver. Starbase can also write directly to the display hardware when operating in an X11 window.
3. The graphics libraries which run on top of Starbase (such as HP-GKS) also run in raw mode.
4. While it is possible to run multiple Starbase applications at one time in raw mode, Starbase does not provide a means to coordinate access to shared resources (for example, the hardware color map). This means that a program can have the hardware colormap changed without its knowledge if other programs are also running in raw mode. The implementation of X11 addresses such resource-sharing issues.

HP Windows/9000 Architecture

The Window Manager accesses the display hardware in creating and moving windows. The diagram below shows how Starbase operates 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. This architecture is similar to the X11 architecture, which also permits Starbase to render directly to the display hardware.

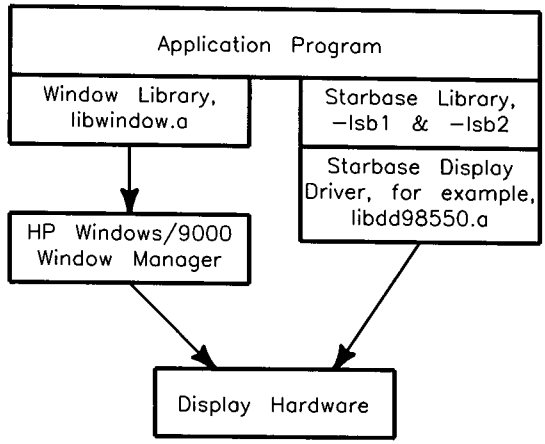


Figure 2-2. Starbase Drivers Directly Access Display Hardware with HP Windows/9000

X Window System Architecture

This section describes elements of the X Window System architecture common to both X10 and X11 and shows how the Starbase-on-X driver (either Xn or SOX11) operates within the X window system.

Client/Server Relationship

The **client** is the program that needs the services and processes of the server in order to run; the **server** is the computer which provides the needed services. The client/server relationship is shown in the following diagram of the X Window System.

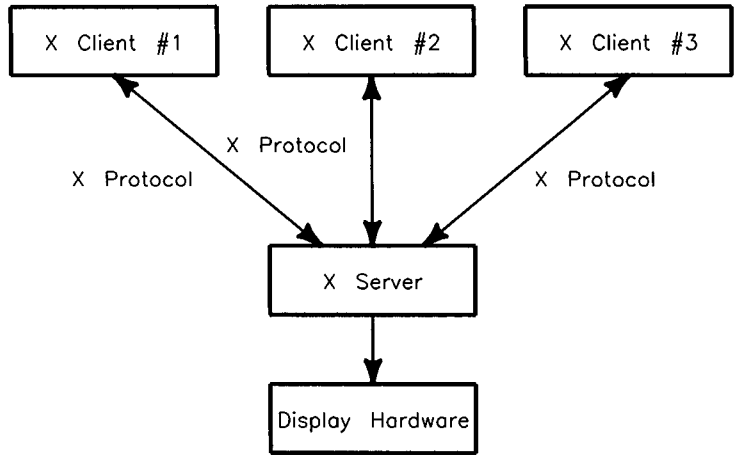


Figure 2-3. X Client/Server Architecture

The main points shown in the previous diagram are:

- Two or more processes are associated with the X window system, the X Display Server (or X Server, for short) and one or more clients. The X Server software controls the X windows themselves and interacts with the user via the keyboard and pointer device. The client programs make requests to the X Server (for example, draw a line, create a window, or get keyboard input) using the X protocol.

- The client program can be run on the same computer as the X server (called **local operation**) or can be run on a different computer than the X server (called **remote operation**). In fact, a client program can be developed to provide **network transparency,** which means that the same program will work both as a local client or a remote client—at run time, you just re-direct the client's input/output to a different server.
- Combinations of local and remote clients can communicate simultaneously to the X server. For example, a remote client can make Xlib calls to render in one window while a local client can make Xlib calls to receive input from a second window.
- When a new display is developed (along with a new X server which supports this display), old client object code will still run because the client/server communications protocol remains unchanged.

The Xlib Library

The Xlib library is used by client programs for X window operations. For example, the Xlib procedure `XDrawDashed` results in communications being sent to the X server to draw a dashed line on the display. There are two versions of Xlib, X10 and X11, associated with the X10 and X11 servers; refer to the appropriate X10 or X11 documentation for a detailed description of the procedures provided by these libraries.

The client box in the previous diagram can be expanded to show how the client's application program resides "on top" of the Xlib library, as shown in the following diagram.

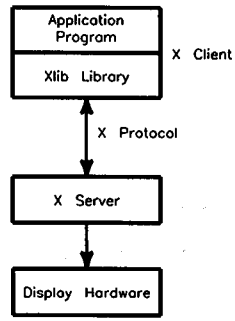


Figure 2–4. X Client Based on Xlib Library

Starbase Operation in an X Window

Starbase can operate in an X window in two ways:

1. Run Starbase in the version of X11 which supports full-functionality Starbase with performance comparable to raw-mode Starbase.
2. Use the Starbase-on-X driver. There are two versions of this driver, an X10-based version (Xn) and an X11-based version (SOX11). The Starbase-on-X driver translates Starbase calls into X protocols for communication with the X server.

The diagram below shows two clients, one which makes only Xlib calls (X Client #1) and one which makes Starbase calls (X Client #2). These Starbase calls are translated into Xlib calls by the Starbase-on-X Driver.

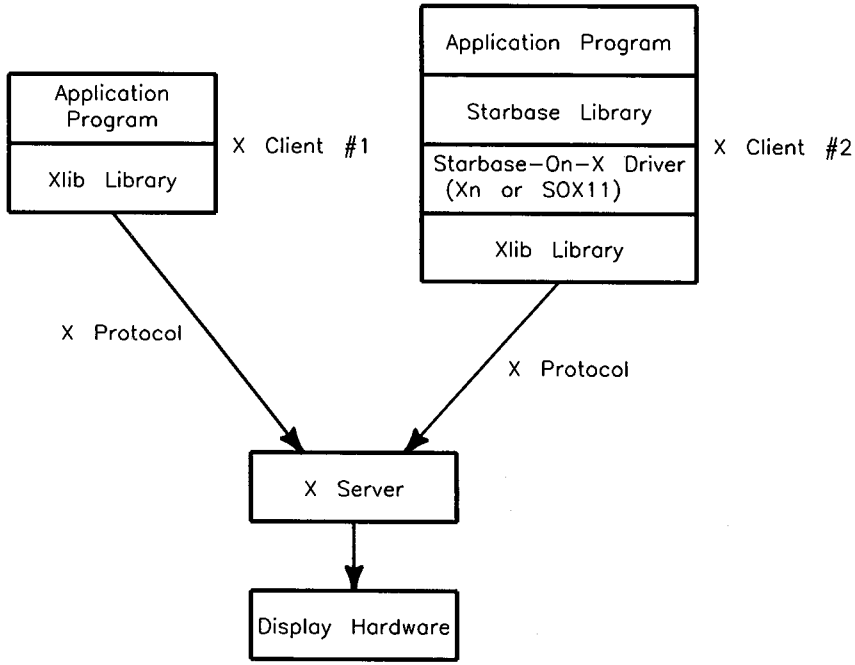


Figure 2–5. Starbase-on-X Driver Architecture

Note the following:

- The Starbase-on-X driver permits client programs to generate Starbase graphics inside an X window. These Starbase procedures are translated by the Starbase-on-X Driver into Xlib calls.
- Several Starbase procedures are NOPs, that is, the Starbase-on-X driver does not attempt to translate them into Xlib calls, because Starbase provides both 2D and 3D graphics primitives while Xlib provides 2D coordinate graphics. Refer to the "X Windows Device Driver" and the "X10/Xn and SOX11" chapters in the *Starbase Device Drivers Library Manual* for a list of the Starbase calls which are not supported by the Starbase-on-X drivers.
- Because the client/server connection can be local or remote, this architecture provides Starbase over the network. However, both the functionality and performance are less than what is provided by Starbase within an X11 window. Starbase works over the network with the Starbase-on-X driver, but without this driver, Starbase only works for local operation.

Parallel Processing with Starbase-on-X

Starbase performance using the Starbase-on-X Driver is less than the performance provided by Starbase rendering directly to an X11 window. This is to be expected because of the overhead associated with translating Starbase calls into Xlib calls, plus the client/server communications overhead. However, the performance impact is tempered because there are two computers working on the task. While the client computer is doing the Starbase-to-X translation, the server computer is (in parallel) rendering the previously received display requests.

X11 and Graphics Architecture

The X11 server shares the display with Starbase. Thus, there are two paths to the display hardware, as shown below:

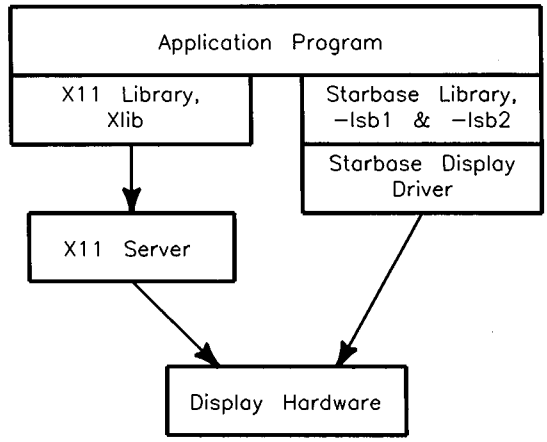


Figure 2-6. Starbase and X11 Architecture

Operating Environments

The X11 server supports three operating environments:

1. Standalone X11 server operation without Starbase being used.
2. Starbase operating in raw (non-window) mode without the X11 server being used.
3. Starbase rendering directly to the X11 window. This requires that the Starbase program be run on the same machine that is running the X11 server. However, the SOX11 driver can be used with the X11 server to achieve remote Starbase.

The following table compares SOX11 Starbase to Starbase rendering directly to an X11 window.

Table 2–1. Starbase Features Using SOX11 Driver and Starbase Display Drivers

Feature	SOX11	Starbase Directly to an X11 Window
Performance	Less than performance achieved by Starbase display drivers.	Full performance, comparable to raw mode Starbase
Functionality	Subset of Starbase	Full functionality, same as raw mode Starbase
Operates Over the Network	Yes	No

3

Using Starbase with the X11 Windows System

Introduction

When using a window system, a computer's working capacity can be divided between several processes, each of which has its own dedicated area of the display for input and/or output. This is perhaps the major benefit of a windowing system. The user can address each window at will, and cause programmatic input/output to be associated with a particular window.

This chapter discusses how Starbase operates with the X11 Windows system. Specifically, this chapter:

- Describes how to take an existing Starbase program and run it in an X11 window. Such a program is called "window-dumb" because it does not call Xlib window procedures. This is in contrast to "window-smart" programs which *do* make Xlib window calls.
- Provides details on window-dumb and window-smart program development to assist you in planning your development activities.

The guidelines in this chapter do not cover the more complex interactions between Xlib and Starbase. For example, these guidelines assume that your X11 server is operating in the server mode necessary to support your graphics program. For example, it is assumed that your server is operating in double-buffer mode if that is what your program expects. A large number of Starbase programs should be able to run in an X11 window by just following the guidelines presented in this chapter. However, if you have difficulty, refer to the other documentation mentioned below.

Setting Up Your "X0screens" File

In the directory `/usr/lib/X11`, there is a file named `X0screens` that contains configuration information required by your X server. It specifies the mode in which the X server operates, the device files to use when communicating with the hardware, the depth(s) of the windows you will be using, and whether to set up color maps for single-buffered or double-buffered operation.

The Operating Modes

There are four different modes in which your X11 server can operate. These are specified by various combinations of parameters in your `X0screens` file. In the following discussion, example lines for the `X0screens` file are given. Assume for the examples below that `/dev/ocrt` specifies your overlay planes, and `/dev/crt` specifies your image planes. Note also that not every depth is valid for every kind of hardware. See the *Starbase Device Drivers Library Manual* for details.

The four operating modes are:

- **Overlay mode:** The X11 server operates only in the overlay planes.

  `/dev/ocrt` *(Uses only overlay planes.)*

- **Image mode:** The X11 server operates only in the image planes. Displays without overlay planes always operate in Image mode.

  `/dev/crt depth 6 depth 8` *(Uses only image planes. Windows are single-buffered, and either six or eight planes deep.)*

  `/dev/crt depth 6 doublebuffer` *(Uses only image planes. Windows are double-buffered, and six planes deep; i.e., 3/3.)*

- **Stacked Screen mode:** The X11 server operates in the overlay planes and the image planes as *two separate screens.* This configuration is similar to a combination of Overlay mode and Image mode. In fact, it is the same as having two separate screens, one with only image planes and the other with only overlay planes. You can switch between the two screens by moving the X window cursor off the left or right edge of the screen.

If there are more than two screens (for example, two physically separate screens, each operating in Stacked Screen mode, for a total of four screens), moving off the left edge moves up the list of screens while moving off the right edge moves down the list. If one end of the list is passed, the screen at the other end is displayed.

`/dev/ocrt` `/dev/crt depth 16 doublebuffer`	*(Uses the overlay planes and the image planes as separate logical devices. Windows are double-buffered, and sixteen planes deep; i.e., 8/8.)*
`/dev/ocrt` `/dev/crt depth 24`	*(Uses the overlay planes and the image planes as separate logical devices. The windows are single-buffered, and twenty-four planes deep.)*

Note that Stacked mode is the only operating mode in which the `XOscreens` file has two lines per physical display device. All the other modes have one line per physical display device.

- **Combined mode:** The X11 server operates in both the overlay planes and image planes simultaneously. This mode was introduced with X11 revision A.01 and was not supported with X11 revision A.00. When an image-plane window is created, a mask of that window is created in the overlay planes and is filled with the transparency color. Thus, the image plane windows and the overlay plane windows *appear* to be in the same set of planes. Image plane and overlay plane windows can obscure each other.

`/dev/ocrt /dev/crt depth 8 doublebuffer`	*(Allows windows in the overlay planes and the image planes. The windows in the image planes are double-buffered, and eight planes deep; i.e., 4/4.)*
`/dev/ocrt /dev/crt depth 8 depth 24`	*(Allows windows in the overlay planes and the image planes. Windows are single-buffered, and either eight or twenty-four planes deep.)*

So Which Mode Should I Use?

In the context of graphics programming, we'll analyze each of the options above and select the best one for the purposes of this example.

- **Overlay mode:** Since, in this mode, the X11 server operates only in the overlay planes, it is obviously not the one to choose for a graphics environment. You need the image planes to do any serious graphics.
- **Image mode:** This mode, because it uses the image planes, would allow advanced graphics. However, in this mode, since *everything* happens in the image planes, the window borders share the hardware color map with the windows. Thus, when the color map for the focus window is downloaded into the hardware, the window borders temporarily assume different colors. Contrast this with Combined mode, in which the only other windows that temporarily assume different colors are those in the image planes. If you can live with the window borders changing colors, Image mode would be satisfactory. (If your hardware has no overlay planes, Image mode is required.)
- **Stacked Screen mode:** This mode is fine if you never want to have the windows in the image planes and the windows in the overlay planes visible simultaneously. For the purpose of our example here, let's say we want to have them both visible at the same time. Therefore, we wouldn't use Stacked mode in this situation.
- **Combined mode:** This mode allows windows in the overlay planes and the image planes. Since they use different color maps, changing the focus to and from a window in the image planes does not cause changes in the window frames' colors; only the colors of the Starbase images themselves. Thus, for graphics purposes, it is perhaps a bit "cleaner" than using Image mode.

Linking the X11 Libraries

The appropriate libraries must be included in the link list in order to get a Starbase-on-X11 program to run. The libraries required for running through the `sox11` driver are as follows:

```
... -lddsox11  -lsb1  -lsb2  -lX11 ...
```

The libraries required for running fully functional Starbase to an X11 window are as follows:

```
... -lXwindow  -lsb1  -lsb2  -lXhp11  -lX11 ...
```

The libraries listed above (as pertains to X11) must be be linked *in the order given.* For example, `-lXwindow` before `-lsb1` and `-lsb2`, and `-lXhp11`, `-lXr11`, and `-lX11` after `-lsb1` and `-lsb2`.

The library `-lXwindow` must be linked to permit the program to run in an X11 window. The libraries `-lXhp11` and `-lX11` must be linked because the library `-lXwindow` is present, even though the application program itself does not call any of the procedures in these libraries.

Running A Window-Dumb Program With X11

Even though a window-dumb program does not make Xlib calls, it can be directed to run within an X11 window. For example, suppose you have a Starbase application that works in "raw mode"; that is, no window system at all. Obviously, such a program would be window-dumb. However, it can remain window-dumb and still run in an X11 window. The X11 window to which the program sends its output, however, must be created before the program runs.

Creating the X11 Graphics Window

To create a window to which you can send Starbase output, use the `xwcreate` command. For example:

```
xwcreate  -geometry 800x600+20+10  -depth 8  GraphWin
```

This command creates a window of width of 800 pixels and a height of 600 pixels. Its upper left corner is at location 20, 10 on the display, and it is eight planes deep. The window is identified by the name `GraphWin`, in the directory specified by `$WMDIR`. For example, if the environment variable `$WMDIR` is set to `/dev/screen` (the default value), `xwcreate` creates the special device file `/dev/screen/GraphWin`, which can be used to access the window (see the manual *Using the X Window System, Version 11* for the `xwcreate`(1) reference page). In this example, we will assume that the device file is in `/dev/screen`. Note that the `$WMDIR` directory is *not* automatically accessed by a user program as it is by `xwcreate`; the *entire pathname* must be used in the gopen statement.

Note Some drivers do not support the use of the "`-r`" parameter when creating a window to be used for Starbase graphical output. These drivers do not support backing store (retained rasters) for windows. See the *Starbase Device Drivers Library Manual* for further details on your particular device.

Opening the X11 Window

Once the destination window has been created, the appropriate information needs to be communicated to the program. This can be accomplished in any of three different ways:

- Hard-coding the desired values into the gopen statement. This has the disadvantage of requiring recompilation to change devices.
- Pass the values for gopen's ⟨*path*⟩ and ⟨*driver*⟩ parameters into the program via argc and argv. This is much better than the above approach, but it has the disadvantage of requiring the values to be specified every time the program is run.
- Pass the values for gopen's ⟨*path*⟩ and ⟨*driver*⟩ parameters into the program via environment variables. This is often better than both the above approaches, because you can set the values once and use them many times, and yet be able to change them without recompiling.

It is the third approach we will take here. By convention, the environment variables' names are SB_OUTDEV, SB_OUTDRIVER, SB_OVDEV, SB_OVDRIVER, SB_INDEV, and SB_INDRIVER for image planes, overlay planes, and input devices, respectively.

To allow the window-dumb program to access the X11 window we created above, merely set SB_OUTDEV to be /dev/screen/GraphWin, and leave the value of SB_OUTDRIVER as it was.

As for the other two parameters to gopen, ⟨*kind*⟩ and ⟨*mode*⟩, they are defined as follows. The ⟨*kind*⟩ parameter is OUTDEV, INDEV, or OUTINDEV, whatever is appropriate for the program. This parameter is unaffected by running the program in an X11 windows environment.

The ⟨*mode*⟩ parameter is also the same as when running in a non-windows environment, *with one important exception.* When opening an X11 window for graphics, you should *never* set the ⟨*mode*⟩ flag to RESET_DEVICE. To initialize the window, use the INIT flag. The RESET_DEVICE flag resets the entire display completely and will interfere with operation of the X11 server.

An Example Program

Compile and run the simple example program below, sending the output to an eight-plane X11 window created via `xwcreate`. All the program does is write the names of eight different colors in the colors named. That is, it writes the word "Red" with red letters, the word "Orange" with orange letters, the word "Yellow" with yellow letters, and so forth. It also does some color map operations, but these will not be explored until the next section. For now, it is just a window-dumb program that should run in a (previously existing) X11 window merely by setting the environment variable `SB_OUTDEV` to an appropriate value.

```
#include <starbase.c.h>                 /* get Starbase definitions */
#include <stdio.h>                      /* get standard I/O functions */
#define  Red            1.0, 0.0, 0.0   /* RGB for red */
#define  Orange         1.0, 0.5, 0.0   /* RGB for orange */
#define  Yellow         1.0, 1.0, 0.0   /* RGB for yellow */
#define  Green          0.0, 1.0, 0.0   /* RGB for green */
#define  Cyan           0.0, 1.0, 1.0   /* RGB for cyan */
#define  Blue           0.0, 0.0, 1.0   /* RGB for blue */
#define  Magenta        1.0, 0.0, 1.0   /* RGB for magenta */
#define  White          1.0, 1.0, 1.0   /* RGB for white */
#define  BoldSerif      8               /* sent to "text_font_index" */
#define  EndOfLine      FALSE           /* sent to "text2d" */
#define  ShowColor(rgb,name)            Entry = rgb_to_index(fildes, rgb);    \
                                        inquire_color_table(fildes, Entry, 1, \
                                          ColorEntry);                        \
                                        printf("[%3d] %s %.2f, %.2f, %.2f\n", \
                                          Entry, name, ColorEntry[0],         \
                                          ColorEntry[1], ColorEntry[2]);
main(argc, argv)                        /* program "NewColorMap.c" */
int     argc;
char    *argv[];
{
    int         fildes;                 /* file descriptor */
    float       ColorEntry[3];          /* color map entry */
    int         Entry;                  /* loop control variable */
```

```
    /*--- do necessary initialization ----------------------------------------*/
    if ((fildes = gopen(getenv("SB_OUTDEV"), OUTDEV, getenv("SB_OUTDRIVER"),
      INIT)) == -1) {
        fprintf(stderr, "%s %s\n", "Error: gopen failed using environment",
          "variables SB_OUTDEV and SB_OUTDRIVER.");
        exit(-1);
    }
    vdc_extent(fildes, 0.0, 0.0, 0.0, 1.25, 1.0, 0.0);
    view_window(fildes, 0.0, 0.0, 1.0, 1.0);
    text_font_index(fildes, BoldSerif);
    character_height(fildes, 0.15);
    character_expansion_factor(fildes, 0.8);
    text_alignment(fildes, TA_CENTER, TA_HALF, 0.0, 0.0);
    if (argc > 1) {                     /* if user supplied random seed... */
        sscanf(argv[1], "%d", &Entry);  /* grab it... */
        srand(Entry);                   /* and set random number generator */
    }
    for (Entry = 1; Entry < 256; Entry++) {     /* randomize color map */
        ColorEntry[0] = (rand() % 101) * 0.01;  /* red */
        ColorEntry[1] = (rand() % 101) * 0.01;  /* green */
        ColorEntry[2] = (rand() % 101) * 0.01;  /* blue */
        define_color_table(fildes, Entry, 1, ColorEntry);
    }
    text_color(fildes, Red);            ShowColor(Red, "Red:    ");
    text2d(fildes, 0.25, 0.8, "Red", WORLD_COORDINATE_TEXT, EndOfLine);
    text_color(fildes, Orange);         ShowColor(Orange, "Orange: ");
    text2d(fildes, 0.25, 0.6, "Orange", WORLD_COORDINATE_TEXT, EndOfLine);
    text_color(fildes, Yellow);         ShowColor(Yellow, "Yellow: ");
    text2d(fildes, 0.25, 0.4, "Yellow", WORLD_COORDINATE_TEXT, EndOfLine);
    text_color(fildes, Green);          ShowColor(Green, "Green:  ");
    text2d(fildes, 0.25, 0.2, "Green", WORLD_COORDINATE_TEXT, EndOfLine);
    text_color(fildes, Cyan);           ShowColor(Cyan, "Cyan:   ");
    text2d(fildes, 0.75, 0.8, "Cyan", WORLD_COORDINATE_TEXT, EndOfLine);
    text_color(fildes, Blue);           ShowColor(Blue, "Blue:   ");
    text2d(fildes, 0.75, 0.6, "Blue", WORLD_COORDINATE_TEXT, EndOfLine);
    text_color(fildes, Magenta);        ShowColor(Magenta, "Magenta:");
    text2d(fildes, 0.75, 0.4, "Magenta", WORLD_COORDINATE_TEXT, EndOfLine);
    text_color(fildes, White);          ShowColor(White, "White:  ");
    text2d(fildes, 0.75, 0.2, "White", WORLD_COORDINATE_TEXT, EndOfLine);
    make_picture_current(fildes);       /* flush pipeline */
    getchar();                          /* maintain graphics resources */
    gclose(fildes);                     /* 'bye */
}
```

Note

Note that because of the color map redefinition, the displayed colors (assuming the window in question is the focus window), although close to what they call themselves, are probably not *exactly* correct. In all likelihood, though, they will be quite close and easily recognizable.

The "Focus Window" and What It Means

In an X Window environment, there is a concept called the "focus window." It is the window with which the keyboard is currently associated. In a typical X environment, for example, when the pointer enters a window, the window border changes and all keystrokes are sent to that window. This is the focus window.

In the context of Starbase, the focus window also comes into play. Suppose you have two Starbase programs running simultaneously, and they both change the default color map to the arrangements they require. Isn't this asking for trouble? It would seem that the program that changed the color map most recently would be the only one with the correct understanding of the current colors. The other program(s) would be using a different one than what they need.

Color Maps and the Focus Window

The above scenario would be true were it not for the `hpwm` window manager, which arbitrates between Starbase programs contending for the graphical resources. In our divergent-color-map scenario, each Starbase program has its own *software* color map, which no other program can affect. However, there is only one *hardware* color map; this is the one used by the display hardware when generating pictures. And here is where the focus window becomes important: *the hardware color map is loaded with the software color map from the current focus window.* Thus, the image in the focus window will have the correct colors, while all others may not. When you change the focus window, a new software color map is downloaded into the hardware color map, and the image in the new focus window suddenly looks right. Meanwhile, the program running in the previous focus window continues to run, but perhaps displaying unexpected colors.

Important Only the `hpwm` window manager does these color map downloading operations.

Now we'll explore more of the features of the example program listed above. If you'll look at the program again, you'll see that it shuffles the color map before it uses it. Then, when selecting a color, requests it via its RGB values instead of its color map index; indeed, since we shuffled the color map, we don't know where the colors are.

Notice also that the random-number seed can be set by passing an integer into the program. This allows the color map to be set up in a different (random) order in different invocations of the program[1]. This capability will turn out useful in this next discussion.

Using the `xwcreate` command as described earlier in the chapter, create two graphics windows. Then, run the example program listed above, passing different random-number seeds to them. The different random-number seeds cause different color map definitions. But since RGB routines were used to select the text colors, "Red" is still printed in red, "Orange" is still printed in orange, and so forth. And here is where the fun starts.

When both windows have the images drawn into them, you'll notice that one (or possibly both) has its words written in the wrong colors. This is not because of a bug in the program, it is an effect of the interaction between multiple software color maps and a single hardware color map: only one of the software color maps can be loaded into the hardware color map at any one time. The choice of software color map to load is determined by the current focus window.

Say, for example, that your two windows are named `W1` and `W2`. Move the mouse back and forth between these two windows. When `W1` is the focus window, it software color map is loaded into the hardware color map, and its color are correct. The colors in `W2`, however, are wrong. Conversely, when `W2` is the focus window, its software color map is loaded into the hardware color map, and its color are correct. And, the colors in `W1` are wrong.

[1] When passing a value into the program as the seed for the random-number generator, you probably wouldn't want to use the value 1, as this is the default. Therefore, passing in 1 would yield the same random number series as not passing in anything.

This process works for as many windows as you have on the screen. If, for example, you have seven Starbase graphics windows visible at once—all with different software color maps—only the focus window's image will appear correct.

Note The Starbase color map is associated with the *program,* not with the *window.* When a program quits, it deallocates all its graphics resources, including the color map. Therefore, the program must still be running in order for color-map downloading to occur (hence the `getchar()`). To see the downloading in action, you will probably want to run the program from two separate `hpterm` windows. Both programs will draw their images, and then wait until you press Return in the `hpterm` windows before terminating.

Window-Dumb X11 Graphical Input

Input devices work similarly. Suppose you have two Starbase programs running simultaneously in two different windows, and they both use graphical input. Where does the input device's information go? Again, it is sent to the window that is currently the focus window.

To illustrate how to use window-dumb input in an X11 window, run the program `Trackmouse.c` (listed in Chapter 6). Before you run it, though, set up the environment variables thus:

```
SB_INDEV=/dev/screen/⟨window⟩
SB_INDRIVER=hp98730
SB_OUTDEV=/dev/screen/⟨window⟩
SB_OUTDRIVER=hp98730
```

where ⟨*window*⟩ is the name of the graphics window in which the program is running, and of course, the driver names should be appropriate for your hardware. It should run as before.

Handling Input Devices

The approach to handling input devices is the same as handling output devices: the gopen ⟨*path*⟩ parameter is passed into the program to permit an input device to be opened in either raw mode or in an X11 window.

The Starbase `hp-hil`, `kbd` and `lkbd` drivers can be used in both raw mode and in an X11 window, based on the gopen ⟨*path*⟩ parameter. Shown below is an example where an HP-HIL device is opened in raw mode and in an X11 window. This example is for an HIL device that is *not* the window system pointer device. An HIL tablet is used; this tablet is identified as the `FIRST_TABLET`. This nomenclature is explained in detail in Chapter 6; suffice it to say that it represents the first tablet on the HP-HIL loop.

```
fildes = gopen("/dev/hil1",                          INDEV, "hp-hil", INIT);   (raw)
fildes = gopen("/dev/screen/win1 FIRST_TABLET", INDEV, "hp-hil", INIT);   (X11)
```

The window system pointer device (typically a mouse) *cannot* be directly opened by the Starbase HP-HIL driver. This is because the X11 server "grabs" this device. However, it is possible to obtain input from the mouse using the `libXwindow` library. This library permits the program to obtain the position of the window system cursor (which is controlled by the mouse). To do so, the

output driver (say, the hp98550 driver) is opened as either an INDEV or OUTINDEV. The ⟨*path*⟩ parameter specifies the window. An example gopen is shown below:

```
fildes = gopen("/dev/screen/win1", OUTINDEV, "hp98550", INIT)
```

The above gopen procedure opens the window as *both* an input device and an output device. When executing Starbase input procedures, the input is returned from the X11 window cursor. If your program attempts to directly open the window system pointer device with the Starbase HP-HIL driver, an error is generated and the open fails.

The Starbase kbd and lkbd drivers can be opened in raw mode and in an X11 window as follows:

```
fildes = gopen("/dev/console", INDEV, "kbd", INIT);        (raw mode open)
fildes = gopen("/dev/ttyq3",   INDEV, "kbd", INIT);        (X11 open)
```

The example special device file /dev/ttyq3 used by the X11 open can be determined by typing tty in the window from which you want to obtain keyboard input. This returns the pty file for the window (for example, /dev/ttyq3). Again, the recommendation is to pass the ⟨*path*⟩ parameter into your program so that you can select the appropriate *path* parameter at runtime. The input devices are added to our example program below. In order for mouse input to be available in raw mode or in an X11 window, its gopen procedure is separated from the display open.

```
#include </usr/include/starbase.c.h>
#include </usr/include/stdio.h>
main(argc, argv)
int     argc;
char    *argv[];
{
    int         outfildes, mousefildes, kbdfildes, tabletfildes;

    outfildes       = gopen(arg[1],OUTDEV,  "hp98550",INIT);
    mousefildes     = gopen(arg[2],INDEV,    arg[3],  INIT);
    kbdfildes       = gopen(arg[4],INDEV,"kbd",INIT);
    tabletfildes    = gopen(arg[5],INDEV,"hp-hil",INIT);
       .
       .
       .
    gclose(outfildes);
    gclose(mousefildes);
    gclose(kbdfildes);
    gclose(tabletfildes);
}
```

The program is linked as follows:

```
cc -o prog1 prog1.o -ldd98550 -lddhil -lddkbd -lXwindow          \
  -lsb1 -lsb2 -lXhp11 -lX11
```

The program is run in raw mode as shown below. The five parameters listed after `prog1` correspond to `arg[1]` through `arg[5]`. This assumes that the special device files are as follows:

Raw display: `/dev/crt`

Mouse: `/dev/hil2`

Keyboard: `/dev/console`

```
prog1 /dev/crt /dev/hil2 hp-hil /dev/console /dev/hil1
```

The program is run in an X11 window using either of the following:

```
prog1 /dev/screen/win1 /dev/screen/win1 hp98550 /dev/ttyq3       \
  "/dev/screen/win1 FIRST_TABLET"
```

4

Graphics Output Operation

Introduction

The sharing of graphics output resources with emphasis on Starbase output operation in an X11 window, and the interaction of Xlib and Starbase procedures are discussed in this chapter.

The concept of programs sharing workstation resources means that a Starbase program running in an X11 window can access all input devices and display resources (for example, the color map) as if the program is running by itself on a non-window (raw mode) display. You can also mix Xlib and Starbase procedure calls in the same program.

This ability to share the input and display resources is a significant benefit because it permits independently developed applications to run simultaneously in different windows without interfering with each other. The ability to mix Xlib and Starbase calls is beneficial because it permits programs to optimize use of workstation resources.

Chapter Organization

This chapter will cover the following topics:

- How the integration of Starbase graphics with X11 affects raw mode Starbase output and Starbase output in HP Windows/9000.
- How Starbase output works in an X11 window.

Raw Mode and HP Windows/9000

This section describes changes to Starbase raw mode operation with the 3.1/6.5 releases. It also describes the changes to HP Windows/9000 operation with the 6.5 release. The major changes are:

1. The process `/usr/lib/grmd`, which is used to manage graphics resources used by Starbase, the X11 server, and HP Windows/9000, is started whenever the display is opened in raw mode (except for the HP 9836A device driver) or whenever the HP Windows/9000 window manager is started, and should be left undisturbed. When the last Starbase display driver is closed, the `grmd` process terminates automatically.

2. Previously, HP Windows/9000 mapped in shared memory for storage of unoptimized fonts, retained window contents, and other data structures in up to five shared memory segments of two megabytes each. Now, this shared memory is allocated as one shared memory segment. The maximum amount of shared memory allocated is controlled by the environment variable `WSHMSPC` (as described in `wmstart`(1) in the *HP Windows/9000 Reference.*

   The previous maximum value of `WSHMSPC` was 10 Mbytes. The new limit is 4 Mbytes, which may be increased by increasing the value of the configurable operating system parameter `shmmax`. The adjustment of this parameter is described in the *HP-UX System Administrator Manual.* Because the entire shared memory maximum is allocated at one time, increasing the `WSHMSPC` variable can increase the swap space that is required even when the window system does not need all the space allowed.

Overview of Starbase Output

The integration of Starbase and X11 provides full-functionality Starbase in an X11 window with performance comparable to raw mode Starbase. Starbase performance in an X11 window is slightly less than raw mode Starbase due to some additional overhead involved with operating within a window.

Full-functionality Starbase in an X11 window means that those graphics libraries built on top of Starbase (for example, HP-GKS) also achieve full functionality. Because Starbase operates in X11 windows, most of the libraries which reside on top of Starbase (for example, HP-GKS) also run in X11 windows. The exception is AGP/DGL, which is not supported in X11 windows (AGP/DGL is, however, supported in raw mode).

Another important capability provided with the integration of Starbase and X11 is that programs can share the output resources without contention. Each program in each X11 window can utilize the display resources, such as the color map, without interfering with other programs using the same resources. This is in contrast to HP Windows/9000, where access to the color map is not arbitrated; any program in any window can change the color map values set by another program.

Xlib Graphics versus Starbase Graphics

The following table assumes that Starbase is operating with its direct display drivers, not the Xn or SOX11 drivers.

Table 4–1. Comparison of Xlib Graphics and Starbase Graphics

Feature	Xlib Graphics	Starbase Graphics
Coordinate Systems	2D, device coordinate only	2D and 3D, supports advanced shading and rendering, virtual device coordinates
Performance	Performance constrained by client/server protocol	Higher than Xlib because Starbase drivers communicate directly to the display hardware.
Vendor Independence	Yes	No
Operates Over Network	Yes	No. The Xn or SOX11 driver can be used but this provides less Starbase functionality and performance.

X11 supports two types of drawables, windows and pixmaps. Starbase can only open X11 windows, not pixmaps. Xlib can access both windows and pixmaps.

Example Interactions Between Starbase and X11

In doing Starbase output in an X11 window, there are several interactions between Starbase operation and X11 window operation. Examples are:

- In order for Starbase double buffering to work, the X11 server should be started in double-buffer mode.
- If Starbase is rendering to an obscured, retained window and the window is resized, the retained rendering of the image is discarded.
- Starbase color map calls can affect the color map associated with an X11 window.

Sharing Display Resources with X11

A program rendering in an X11 window controls rendering through the following two groups of control data:

- Display-control data
- Drawing-control data

Display-Control Data

The **display-control data** affects what is seen on the entire display. An example is the **display-enable mask.** The display-enable mask affects the visual appearance of your program's output and of the entire display. Display-control data affects what you see on the display because only one set of supporting hardware is provided for each display-control data item.

The bitmapped displays support one hardware color map. Each program can have its own software color map. When a program's software color map is downloaded into hardware, the colors for that program will be shown as expected. Other programs in other windows will not have correct colors unless their color maps are identical to the one currently in hardware. Display-control data does not affect what is written into the frame buffer; it only affects what is seen on the display.

Typical display-control data items for Starbase and Xlib are shown in the following table:

Table 4–2. Starbase and Xlib Display Control Data

Display Control Data	Starbase Usage	Xlib Usage
Display Enable	Specified for each open	Not supported; all planes enabled by Xlib
Plane Blinking Control	Specified for each open	Not supported by Xlib
Colormap	Specified for each open	Xlib permits creation of multiple software color maps
Double Buffer Control	Starbase can specify for each window opened. The server, however, should be started in double-buffer mode.	Server must be started in desired double-buffer mode

The only display-control data that Xlib can change is the color map. Double-buffer control is specified when the X11 server is started up, not by Xlib calls. A set of display-control data is associated with each Starbase open of an X11 window.

Drawing-Control Data

In contrast to the display-control data, the **drawing-control data** affects what is *written* to the frame buffer. It only affects the visual appearance of the contents of the window which is currently being rendered. Typical drawing control data includes the following:

- Replacement rule
- Write-enable control. The write-enable mask can be set directly, using Starbase `write_enable` or Xlib `XChangeGC`, or indirectly, using such calls as Starbase's `dbuffer_switch`.
- Line color, type, and width

Whenever a program using Xlib or Starbase renders to a window, that program's drawing control data is used.

When using both Xlib and Starbase within the same program to draw to a window, you may choose to make the Xlib and Starbase drawing-control data the same using the appropriate Xlib and Starbase procedure calls. This permits rendering to be done in a consistent manner for both Xlib and Starbase. For example, to set the default replacement rule to `EXCLUSIVE-OR` for both Xlib and Starbase, use `XSetFunction` to set the X11 value and `drawing_mode` to set the Starbase value.

Note that the default drawing-control data is, for the most part, the same for Xlib and Starbase. For example, the replacement rule is source, the write enable mask is all ones, the line type is solid, the foreground color is white and the background color is black. An application that uses only the default values may not require any action at all to make Xlib and Starbase output consistent. An application that changes the drawing-control data needs to decide whether to make identical changes to both Xlib and Starbase or if different values are acceptable.

Display-Control Policy

The **display control policy,** specifies how the display-control data is handled. Remember, the display-control data affects what is seen on the screen but not the actual data written into the frame buffer.

The condition for the display-control data of a window to be downloaded into hardware are specified by the display-control policy. The **display-control focus window** is the window where you want graphics output to appear "true" (perhaps at the expense of the appearance of graphics in other windows). The display hardware registers will always contain the display-control data for the current display-control focus window. In essence, the display-control focus window "owns" the display-control hardware. As the display-control focus window changes from one window to another, the new window's display-control data is downloaded into the hardware.

The display-control focus window is not to be confused with an **input focus window;** the input focus window is described in Chapter 6. While the display-control focus window and the input focus window may be the same window (for example, both may follow the mouse position), they can also be independently controlled.

Rendering still occurs in windows that are not the display control focus window so data rendered to these windows is not lost. Changes made to display-control

data while a window is not the display-control focus window will be downloaded and displayed when the window becomes the display-control focus window.

Selection of the Display-Control Focus Window

The `hpwm` window manager implements the display-control policy. This window manager, developed by Hewlett-Packard, has the display-control policy built in. Refer to *Using the X Window System, Version 11* for a description.

At X11 startup time, you can specify that the display-control policy should be tied to the position of the window system pointer device (typically a mouse). The display-control policy is based on the window where the window cursor currently resides; this window will have its display-contol data downloaded into display hardware. Some examples are:

- When the cursor leaves one window and enters another window that has been opened by Starbase, the new window's Starbase display-control data is downloaded into the display hardware.
- When the cursor leaves a window and enters a non-Starbase (that is, X only) window, the display-control data for that window is downloaded.
- When the cursor leaves the window and goes to the "desktop", control of the display control data reverts to the X11 server because the "desktop" is actually the X11 **root window.**

Effects on Double-Buffering Operation

The effect of the display-control focus window on double buffering operation is perhaps the most noticeable. When a program in one window is using double buffering while a program in another window is not (that is, it's using all available planes to render its image), and if the window in which double buffering is occurring is not the display-control focus, there will be some flashing of the image as the clearing and drawing associated with double buffering operation occurs. When the double-buffered window is the display-control focus window, the image in the non-double-buffered window will flash when the display-enable changes associated with double buffering are made.

X11 Server Operating Modes

Displays that have only image planes (no overlay planes) support only one server operating mode while displays that have overlay planes support from two to four server modes. The server operating mode affects the Starbase graphics capabilities that are supported within an X11 window.

The four X11 server operating modes are:

- **Overlay Mode:** The X11 server operates only in the *overlay planes.*
- **Image Mode:** The X11 server operates only in the *image planes.* Displays without overlay planes always operate in image mode.
- **Stacked Screen Mode:** The X11 server operates in the overlay planes and the image planes as 2 separate screens. This configuration is similar to a combination of overlay mode and image mode. In fact, it is the same as having two separate screens, one with only image planes and the other with only overlay planes. You can switch between the two screens by moving the X window cursor off the left or right edge of the screen.

  When there are more than two screens (for example, two physically separate screens, each operating in stacked screen mode, for a total of four screens), moving off the left edge moves up the list of screens while moving off the right edge moves down the list. When one end of the list is passed, the screen at the other end is displayed.
- **Combined Mode:** The X11 server operates in both the overlay planes and image planes simultaneously. When an image plane window is created, a mask of that window is created in the overlay planes and is filled with the transparency color. Thus, the image plane windows and the overlay plane windows appear to be in the same set of planes. Image plane and overlay plane windows can obscure each other.

Selection of the operating mode is controlled by the `Xnscreens` file at X11 server startup. Refer to *Programming With Xlib, Version 11* for a description of this file. The following table summarizes the key features of the four server modes.

Table 4–3. Features of the Four X11 Server Modes

Feature	Overlay Mode	Image Mode	Stacked Screen Mode	Combined Mode
Planes Where X11 Server Operates	Overlay planes	Image planes	Two screens, one in overlay planes, one in image planes	One screen combining image and overlay planes.
Location of Root Window	Overlay planes	Image planes	Overlay planes for overlay plane screen, image planes for image plane screen.	Overlay planes
Planes Accessible by Starbase	Overlay plane windows can be opened by Starbase, image planes can be opened in raw mode.	Image plane windows can be opened by Starbase, raw mode Starbase not supported in overlay planes.	Image and overlay plane windows can be opened by Starbase, raw mode Starbase is not supported in any planes.	Any window can be opened by Starbase, raw mode Starbase is not supported in any planes.
Double Buffer Support	No, in overlay plane windows; yes, in image planes in raw mode.	Yes, for image plane windows	Yes, for image plane windows only	Yes, for image plane windows only
Planes Supportd by Backing Store	All overlay planes supported for Xlib and Starbase.	All image planes supported for Xlib, 8 planes maximum for Starbase	Overlay—all supported for Xlib and Starbase.	Image—all supported for Xlib, 8 planes max for Starbase.

Determining the Server Operating Mode

You can use the procedure `XHPGetServerMode` to find the X11 server operating mode if needed. See *Programming With Xlib, Version 11* for details on this procedure.

Supported Visual Classes

The X11 Window System developed by MIT supports six visual classes which determine how color is used in X11 windows. Four of these are supported by the X11 server, as shown in following table.

Table 4-4. Visual Classes Supported by X11

Visual Class	Supported by X11
`PseudoColor`	Yes
`DirectColor`	Yes
`StaticGray`	Yes
`GrayScale`	Yes
`StaticColor`	No
`TrueColor`	No

Refer to *Programming With Xlib, Version 11* for a description of these visual classes.

The following table shows the depths and visual classes supported by each display operating in the four server modes.

Table 4–5. Supported Visuals in The Four X11 Server Modes

Displays	Overlay Mode	Image Mode	Stacked Screen Mode	Combined Mode
300 Hi-Res Dis-plays		*318M:* 1 plane (SG) with B/S *98544A:* 1 plane (SG) with B/S *98545A:* 4 planes (PC) with B/S 2/2(PC) with B/S *98547A:* 6 planes (PC) with B/S 3/3 planes(PC) with B/S		
300 Medium Res Dis-plays		*98542:* 1 plane (SG) with B/S *98542A:* 4 plane (PC) with B/S 2/2(PC) with B/S		
98548A		1 plane(GS) with B/S		

Table 4–5. Supported Visuals in The Four X11 Server Modes Continued

Displays	Overlay Mode	Image Mode	Stacked Screen Mode	Combined Mode
98549 and 319C		6 plane(PC) with B/S 3/3 (PC) with B/S		
98550A	2 overlay planes (SG) (3 colors plus transparency) with B/S	8 planes(PC) with B/S 4/4(PC) with B/S	2 overlay planes (SG) with B/S 8 image planes (PC) with B/S 4/4 (PC) with B/S	
98721A	3 overlay planes (PC, 8 colors) with B/S	4/4 image planes (PC) with B/S 8 image planes (PC) with B/S 8/8 image planes (PC) with X B/S 12/12 image planes (DC) with X B/S 24 image planes (DC) with X B/S	3 overlay planes (PC, 8 colors) with B/S 4/4 image planes (PC) with B/S 8 image planes (PC) with B/S 8/8 image planes (PC) with X B/S 12/12 image planes (DC) with X B/S 24 image planes (DC) with X B/S	

Table 4–5. Supported Visuals in The Four X11 Server Modes Continued

Displays	Overlay Mode	Image Mode	Stacked Screen Mode	Combined Mode
98732A	3 overlay planes (PC, 8 colors) with B/S 4 overlay planes (PC, 16 colors) with B/S	4/4 image planes (PC) with B/S 8 image planes (PC) with B/S 8/8 image planes (PC) with X B/S 12/12 image planes (DC) with X B/S 24 image planes (DC) with X B/S	3 overlay planes (PC, 8 colors) with B/S 4 overlay planes (PC, 16 colors) with B/S 4/4 image planes (PC) with B/S 8 image planes (PC) with B/S 8/8 image planes (PC) with X B/S 12/12 image planes (DC) with X B/S 24 image planes (DC) with X B/S	3 overlay planes (PC, 7 colors) with B/S 4 overlay planes (PC, 15 colors) with B/S 4/4 image planes (PC) with B/S 8 image planes (PC) with B/S 8/8 image planes (PC) with X B/S 12/12 image planes (DC) with X B/S 24 image planes (DC) with X B/S

Definitions for the previous table:

- "B/S" indicates that Backing Store is supported for both Xlib and Starbase.
- "X B/S" indicates that Backing Store is supported only for Xlib.
- "n/n" indicates that double buffering is supported, with n planes in each buffer.
- "PC" indicates that `PseudoColor` is supported.
- "GS" indicates that `GrayScale` is supported.
- "SG" indicates that `StaticGray` is supported.
- "DC" indicates that `DirectColor` is supported

Selecting the Server Operating Mode

The following guidelines are provided to help you select the server operating mode:

- Combined Mode is recommended for the HP 98730. This mode offers the greatest choice of visuals and permits terminal emulator windows to be placed in the overlay planes so they won't interfere with graphics in the image planes.
- Image Mode or Stacked Screen Mode is recommended for the HP 98550. The eight image planes provide more capability than the two overlay planes.

Guidelines for Visuals

As discussed in the document *Using the X Window System, Version 11,* the `XOscreens` file is used to specify the server operating mode and the visual(s). Several guidelines that should be followed are:

1. In Combined Mode on the 98730A with a 24-plane system operating in double-buffered mode, the following combinations of visuals are supported:
   - 4/4 and 12/12
   - 8/8 and 12/12

   4/4 and 8/8 together are not supported.
2. In Combined Mode on the 98730A with a 24-plane system that is not operating in double-buffered mode, 8- and 24-plane visuals can be enabled simultaneously.
3. In Stacked Screens Mode with a 24-plane system operating in double-buffered mode, only one visual is supported at a time (4/4, 8/8 or 12/12).
4. In Stacked Screens Mode with a 24-plane system that is not operating in double-buffered mode, 8- or 24-plane visuals can be enabled, but not at the same time.

Example: Specifying "Combined Mode" and Creating Windows

This section discusses how the server mode is specified and how windows are subsequently created. Combined Mode is used as an example since this is the most

complex (and capable!) mode. The following example `XOscreens` file specifies a Combined Mode server:

```
/dev/ocrt  /dev/crt  depth 8  depth 24  doublebuffer
```

This specifies the following:

- Operation is Combined Mode since `/dev/ocrt` and `/dev/crt` are provided in the same line.
- The server will supports both 4/4 double-buffered windows (eight planes total) and 12/12 double-buffered windows (24 planes total) in the image planes.
- The depth of the server in the overlay planes is controlled by the `/dev/ocrt` special device file. This special device file can specify either 3-plane or 4-plane overlay operation (refer to the *Starbase Device Drivers Library Manual* for information on the special device files).

When you create a window with the `XCreateWindow` procedure, the depth parameter must correspond to a depth supported by the server. In double-buffer mode, the depth parameter is set equal to the depth of *each buffer* (for example, 12), not the total number of planes.

If `/dev/ocrt` specifies three-plane operation and the `XCreateWindow` depth parameter specifies a three-plane window, the window will be created in the overlay planes. Likewise, for the above `XOscreens` file, a window created with a depth equal to 12 is automatically placed in the image planes.

Additionally, the correct visual must be specified. Thus, to get a 12-bit window, the depth must be 12 and the visual must be the visual returned from `XGetVisualInfo` corresponding to the 12-bit, "`DirectColor`" visual.

If a window is to be created in a visual different from that of the window's parent, the client *must* supply a colormap (`CWColormap` attribute) that is of the visual type requested and a border pixel or pixmap (`CWBorderPixel` or `CWBorderPixmap` attribute.)

If a depth is requested (for example, 2) in `XWindowCreate` that does not correspond to the server operating mode, an error is generated.

Now let's look at another example `XOscreens` file:

```
/dev/ocrt  /dev/crt  depth 8  doublebuffer
```

Assume that /dev/ocrt specifies four-plane operation in the overlay planes. The above XOscreens file specifies 4/4 double buffering in the image planes. If you were to then use XCreateWindow to create a window with a depth of four, it is ambiguous whether this window should be placed in the overlay planes or the image planes. Therefore, this XOscreens file is *not allowed* if /dev/ocrt specifies a four-plane system; if /dev/ocrt specifies a three-plane system, this will work.

As Table 4–5 shows, there is a particular visual class (for example, DirectColor) associated with each window depth. Thus, if you are using the 98730A display in Combined Mode and you create a 24-plane window, its visual class is always DirectColor.

Guidelines for Portability

For portability across different X11 window operating environments on HP workstations, it is recommended that application programs do the following:

1. Use XHPGetServerMode to determine which server mode is active.
2. Use XGetVisualInfo to obtain a list of visual structures that match the desired visual attributes. Alternatively, XMatchVisualInfo can be used to obtain the visual information that matches the desired depth and class.
3. Once a visual is found that matches the needs of your application, you can use XCreateWindow to create a window of the desired depth and visual class. If this visual class is not the default visual class, you will have to create a color map for the window.

Transparency Index

This section describes the transparency index used in the overlay planes.

HP 98720 Display

Even though there are three overlay planes, the HP 98720 display only supports *seven* (not eight) colors in the overlay planes. One color is used as the transparency color to see through to the image planes; color index 7 or 15,depending on the overlay depth. Anywhere this color index is written into the overlay planes, the values in the image planes, and not those in the overlay planes, are visible.

HP 98730 Display

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 colors 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 98550 Display:

The HP 98550 display has two overlay planes. Only three colors are available because index 0 is a transparent color.

Supported Starbase Drivers

The following table shows which Starbase drivers can be used in the overlay and image planes for the different server operating modes. If a particular driver is not listed for a particular display and server operating mode, then the driver is not supported for that configuration.

Table 4–6. Starbase Drivers Supported in the Four X11 Server Modes

Displays	Overlay Mode	Image Mode	Stacked Screen Mode	Combined Mode
300 Hi-Res Dis-plays		hp300h		
300 Med. Res Dis-plays		hp300l		
98548		98550		
98549 and 319C		98550		
98550 and 98556	Overlay plane window can be opened by the 98550 or 98556 drivers. The image planes can be opened by the 98550 or 98556 driver in raw mode.	Image plane window can be opened by the 98550 or 98556 driver. Raw mode open cannot be done by any driver.	Overlay or image plane windows can be opened by the 98550 or 98556 drivers.	

Table 4-6. Starbase Drivers Supported in the Four X11 Server Modes Continued

Displays	Overlay Mode	Image Mode	Stacked Screen Mode	Combined Mode
98720 (Note 2)	Overlay plane window can be opened by the 98720 driver. The image planes can be opened by the 98720 or 98721 drivers in raw mode. This is the only server mode where the 98721 driver can be used (but only in raw mode).	Image plane window can only be opened by the 98720 driver. Raw mode open cannot be done by any driver.	Either overlay or image can be opened by the 98720 driver. Raw mode open cannot be done by any driver.	
98730	Overlay plane windows can be opened by the 98730 driver but not by the 98731 driver. The image planes can be opened by the 98730 or 98731 drivers in raw mode.	Image plane windows can be opened by the 98730 or 98731 drivers. Raw mode open cannot be done by any driver.	Window can only be opened by the 98730 driver. Raw mode open cannot be done by any driver.	Window can be opened by the 98730 (image or overlay planes) or the 98731 image planes only). Raw mode open cannot be done by any driver.

Notes on the X11 Server Modes

1. 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.
2. The HP 98720w driver, which was developed to support HP Windows/9000, cannot be used with any server mode (neither within a window nor in raw mode).

Use of Starbase Graphics Accelerators

The X11 server does not use the graphics accelerators for its window and rendering operations. Therefore, Xlib performance is comparable with or without the HP 98556A, HP 98721A or HP 98732A graphics accelerators installed. Starbase performance in an X11 window, however, benefits greatly from usage of the HP 98556A or HP 98732A graphics accelerators. The HP 98721A accelerator is not supported by Starbase within an X11 window; it can, however, be used to accelerate raw mode graphics in the image planes with the X11 server in overlay mode. You can create the appearance of accelerated graphics in an X11 window on the HP 98721A by appropriately positioning the overlay plane window borders over a HP 98721A-generated image in the image planes.

Opens Done with Accelerator Drivers

This section describes the number of Starbase opens that can be done in an X11 window using the HP 98732A and HP 98556A accelerator drivers. You can mix and match open commands using accelerated drivers and open commands using non-accelerated drivers, subject to limits on the number of accelerated windows as discussed below.

HP 98556A Driver:

The number of windows that can be opened with the 98556A accelerator driver is:

1. A maximum of 31 opens can be active simultaneously using the HP 98556A accelerated driver. This limits accelerated graphics to a maximum of 31 windows. If some of the windows contain multiple open commands using the HP 98556A driver, the limit is correspondingly lower.
2. When 31 open commands of the HP 98556A are currently active and another open of the HP 98556A is done, 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.
3. In addition to the 31 open commands that can be done with the accelerated driver, any number of other windows may be opened with the unaccelerated HP 98550 Starbase device driver.

HP 98732A Device Driver:

The number of windows that can be opened with the HP 98732A accelerated driver are:

1. The HP 98732A 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 the *Starbase Device Drivers Library 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. Therefore, it is recommended 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.

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 98732A. 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.

X11 Color Map Control

This section assumes that you are familiar with what a color map is and the Starbase and Xlib procedures which affect the color map. For a description of color maps and how Starbase controls the color map, refer to the chapter "Color Graphics" in *Starbase Graphics Techniques* For a description of Xlib control of the color map, refer to *Programming With Xlib, Version 11.*

The `hpwm` window manager implements the display-control policy. This window manager automatically downloads each window's associated color map into the display hardware when the window cursor enters the window. This policy applies to operation before and after an X11 window is opened by Starbase.

The **color map policy** is an additional mechanism needed to control the interaction of Starbase and Xlib color map procedures.

Hardware and Software Color Maps

It is necessary to distinguish between hardware color maps and software color maps. The **hardware color map** provides the actual physical translation between a pixel value in the frame buffer and the color generated on the screen. There is only one hardware color map for each display. The **software color map** is the program's representation of the color map. There can be numerous software color maps. Colormap calls always affect the software color map; they may or may not affect the hardware color map.

X11 versus Starbase Color Map Modes

Starbase supports the three color map modes listed below; refer to the *Starbase Graphics Techniques* for information on these modes.

- `CMAP_NORMAL`
- `CMAP_FULL`
- `CMAP_MONOTONIC`

The following table shows how the Starbase color map modes and X11 visual classes are related:

Table 4–7. X11 and Starbase Color Map Modes

X11 Mode	Description	Usable Starbase Color Map Modes (default = *)

`PseudoColor`	Definable RGB triples.	`CMAP_NORMAL*`
		`CMAP_FULL`
		`CMAP_MONOTONIC`

`GrayScale` (see notes)	Like PseudoColor but R=G=B	`CMAP_MONOTONIC` `CMAP_NORMAL*`
`StaticGray`	Like GrayScale but pre-defined RGB values	`CMAP_NORMAL*` `CMAP_MONOTONIC`
`DirectColor`	Three independent RGB values	`CMAP_FULL*`

Notes on Color Map Modes

X11 mode using `GrayScale` should keep operation gray scale, but Starbase can override this.

Starbase Use of Color Maps

In a non-window (raw mode) application, a Starbase program has complete control of the hardware color map. The program can set and change the color map values at any time. The ability of a program to control the color map is preserved to support porting of raw mode programs to the X11 window system without requiring source code changes.

Starbase Interactions with the X11 Color Maps

There are three phases of Starbase interaction with the X11 color maps:

- Phase 1: When an X11 window is first created.
- Phase 2: When the X11 window is opened by Starbase.
- Phase 3: After the X11 window is closed by Starbase.

There are two cases of interest which depend on the `mode` parameter of `gopen` that you provide when you open the X11 window: `INIT` being present, and `INIT` being absent. The following sections decribe these two cases.

INIT Present

The following diagram shows the 3 phases:

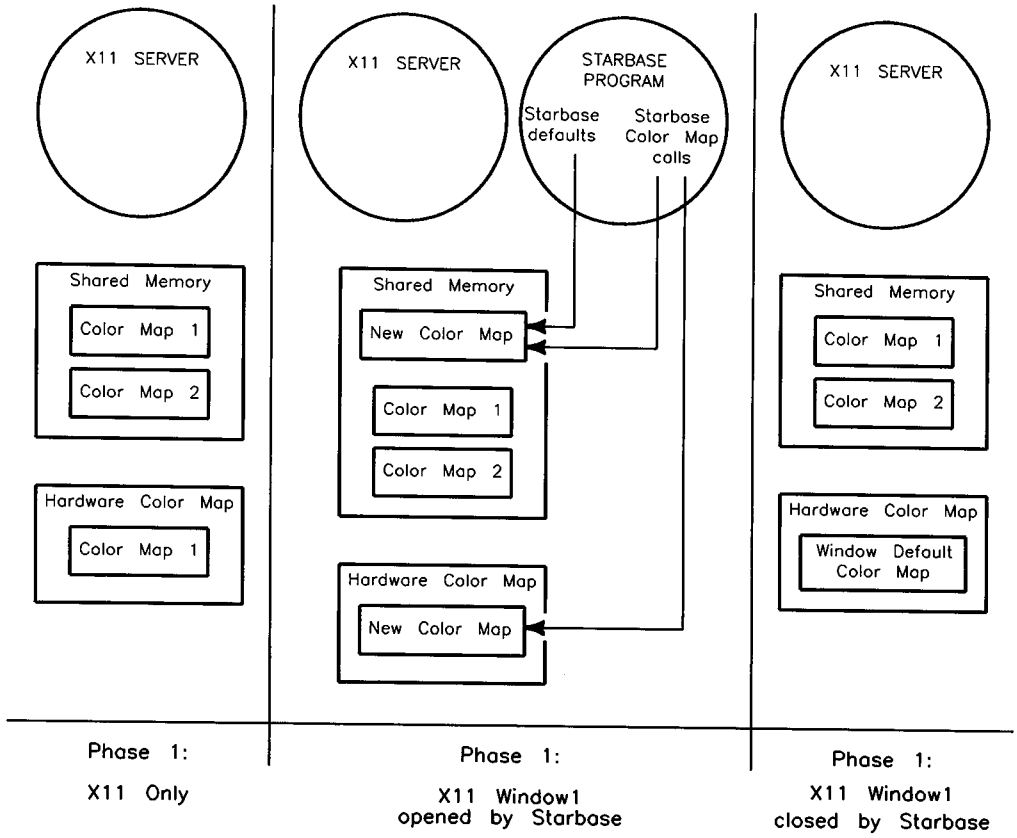


Figure 4–1. Color Map Control with `INIT` Present

All Starbase and Xlib software color maps are stored in shared memory controlled by the Graphics Resource Manager (GRM). In Phase 1, assume that the shared memory already contains two color maps for two windows. Assume also that the window cursor is in Window 1, so that Color Map 1 is automatically downloaded into the hardware color map.

When Window 1 is opened by Starbase during Phase 2, a new X11 color map (called *New Color Map*) is created for the X11 window. New Color Map has the following characteristics:

1. It has the same visual class that Window 1 currently has. For example, when Window 1 has the visual class PseudoColor with depth equal to 8, New Color Map also has the visual class PseudoColor with depth equal to 8.
2. Because `INIT` is present, New Color Map is initialized with the Starbase default color map values for the particular display (see the *Starbase Device Drivers Library Manual* for these default initialization values).
3. When the window cursor is in Window 1, New Color Map is automatically downloaded into the hardware color map as soon as it is created. When the window cursor is in another window, New Color Map is still created and initialized in shared memory but it is not downloaded into the hardware color map until the window becomes the display-control focus window.

Starbase color map operation is as follows during Phase 2:

1. Starbase color map calls affect only New Color Map, not any of the other color maps; Xlib calls can be used to change the other color maps.
2. When New Color Map has been downloaded into the hardware color map, Starbase color map calls modify both shared memory's New Color Map and hardware's copy of New Color Map. When New Color Map is not downloaded into hardware, Starbase color map calls affect only shared memory's copy of the color map.
3. When an Xlib call selects a different color map for Window 1 (for example, Color Map 1) than the color map Starbase "believes" is the current color map then:
    - You can detect a color map change using the `ColormapNotify` event by passing `ColormapChangeMask` to `XSelectInput`.
    - You can use the Starbase gescape `READ_COLOR_MAP` to modify the newly selected color map. This gescape causes Starbase to view this newly selected Color Map 1 as its current color map.

When you close the window in Phase 3, the color map created when the window was opened (New Color Map) is destroyed; Color Map 1 and Color Map 2 continue to exist.

INIT Absent

The following diagram shows the three phases of interest:

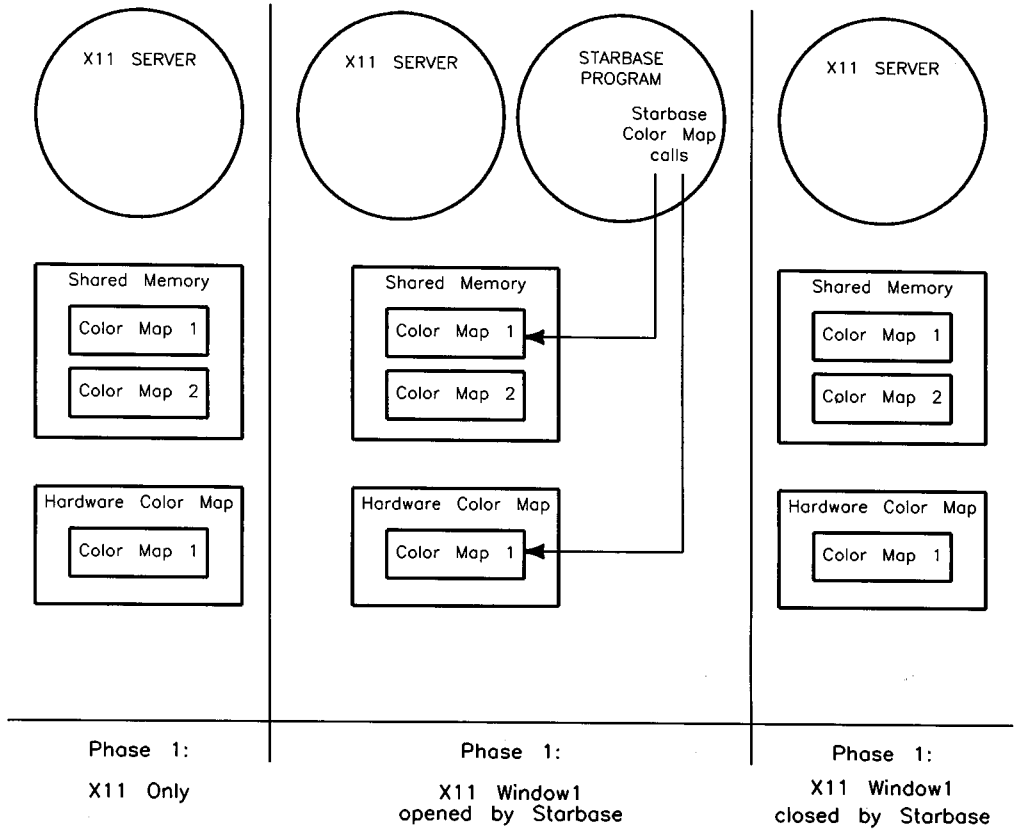


Figure 4–2. Color Map Control with `INIT` Absent

In Phase 1, assume shared memory already contains two color maps for two windows, and that the window cursor is in Window 1, so that Color Map 1 is automatically downloaded into the hardware color map.

In Phase 2, after the Starbase open of Window 1 is done with `INIT` absent, Starbase will receive the ID of Window 1's color map (Color Map 1) and begin using it. The Starbase default values will not be loaded into Color Map 1 because `INIT` is false. However, subsequent Starbase color map calls will affect the contents of Color Map 1. When Color Map 1 is installed in the hardware

color map, Starbase color map calls affect the hardware color map. Starbase will also respect read-only attributes of cells in the software color map and will not change them.

In Phase 3, after the window is closed, color map operation is again under control of the Xlib library.

Multiple Processes Opening a Single Window

Assume two processes (Process 1 and Process 2) open a single window, and that Process 1 opens the window with `INIT` followed by Process 2 opening the window without `INIT`. Consider the following sequence:

1. Process 1 opens with `INIT`: A new color map (New Color Map) is created.
2. Process 2 opens without `INIT`: Process 2 inherits the window's current color map (New Color Map) as its color map.
3. Process 1 closes the window: At this point, Process 1's New Color Map is disassociated from the window. While the color map will continue to exist in shared memory and can be accessed by Process 2, the window manager will have no knowledge of its existence and thus will no longer be able to download the color map into the hardware color map.
4. Process 2 can detect color map changes as described previously, namely using the `ColormapNotify` event by passing `ColormapChangeMask` to `XSelectInput`. Process 2 can re-associate New Color Map with the window using `XSetWindowColormap`. Process 2 can inherit the current color map using `READ_COLOR_MAP`. This will permit Process 2 to access a color map that the window manager can download into the hardware color map.

Non-interacting Color Maps

The Xlib and Starbase color map calls only interact when Starbase opens an X11 window. For example, when the X11 server is operating in the HP 98732A overlay planes and your program does a raw mode open of the image planes, there is no color map interaction because the overlay planes and image planes each have their own separate color maps. If the X11 server and Starbase are used on the HP 98547A display (where both Xlib and Starbase output go to the image planes), both Xlib and Starbase affect the single hardware color map.

X11 and Starbase Color Map Cooperation

Application developers are encouraged to design color map cooperation into their application programs. Suggestions for implementing color map cooperation are:

- Use the X11 color maps and color map calls whenever possible.
- Initialize a few identical entries in all color maps and use these colors for borders, foreground/background, text, and any other commonly displayed information.
- Minimize the number of different X color maps that are used.
- Start up X11 using the default Starbase color map when running window-dumb Starbase applications that do not alter the color map. You can use the `Xinitcolormap` utility to initialize the color map immediately after starting the server.

X11 Double-Buffering Operation

This section describes double buffering operation with the X11 server. For a description of how Starbase double buffering works in raw mode, refer to the chapter "Frame Buffer Control and Raster Operations" in the *Starbase Graphics Techniques* manual. For a description of double buffering support provided by the X11 server, refer to *Programming With Xlib, Version 11.*

The buffer sizes in double-buffer mode are denoted as 2/2, 4/4, etc. For example, 4/4 means that there are four planes in each buffer for a total of eight planes. In effect, the device contains two separate frame buffers, each with four planes.

X11 Support of Double Buffer Mode

As described in *Programming With Xlib, Version 11,* the X11 server can be started in double-buffer mode. Once the X11 server is started, the double-buffer mode cannot be enabled or disabled. Likewise, the size of each of the two buffers is fixed at server startup time and cannot be changed after starting. Double buffering of the X11 server is controlled by the `Xnscreens` file, where *n* denotes the display number (0 is used in our discussions). The following example contents of the `X0screens` file starts the X11 server in 4/4 double-buffer mode.

```
/dev/crt  depth 8  doublebuffer
```

The X11 server operates in double-buffer mode as follows:

- Instead of selectively rendering in one buffer or the other, Xlib renders the same data simultaneously to both buffers. Thus, as the displayed buffer is toggled between buffers, the appearance of the X11 window borders and Xlib rendering is unchanged.
- The X11 default color map is initialized so it has the same colors for each of the two buffers. Subsequent Xlib color map calls automatically change the color map for both buffers.
- The depth of visual supported for clients is the depth of one buffer.

Starbase Support of Double Buffering

When the X11 server starts in double-buffer mode, Starbase programs may operate in double-buffer mode. Use the Starbase call `inquire_display_mode` to determine whether a Starbase program is operating in single- or double-buffer mode, the size of each buffer, and the active buffer. Refer to the *Starbase Reference* for details on this procedure.

When the X11 server is operating in single-buffer mode, the Starbase application should operate in single-buffer mode.

Starbase programs that do double buffering will "inherit" the double-buffer environment that the X11 server uses at startup. The following cases illustrate how X11 and Starbase double buffering interact:

- When the X11 server starts in 4/4 double-buffer mode on an eight-plane system, a Starbase application can operate in 4/4 double-buffer mode.
- When the Starbase program requests less than 4/4 double-buffer mode (for example, 2/2 double buffering), it will only have access to the two lower planes of each four-plane buffer.
- When a Starbase program requests larger buffers (for example, 8/8 double buffering) than is provided by the X11 server (for example, 4/4 double buffering), Starbase will be limited to 4/4 double-buffer mode.
- When a Starbase program enables double buffering with a server that is not operating in double-buffer mode, the Starbase program will be allowed to operate in double-buffer mode. However, because the X11 server is not operating in double-buffer mode, it will use the entire display depth for its window borders and Xlib rendering. As the buffers are alternately displayed by Starbase, partial segments of the display data rendered by Xlib will flash on the screen. Because this may be visually disruptive, Starbase programs should only use double-buffer mode when the X11 server is started in double-buffer mode.

When the X11 server is started in double-buffer mode, your program can open a window and specify double-buffer operation using the Starbase `double_buffer` procedure. This permits your program to write to the non-displayed buffer while the other buffer is displayed. Note that switching between buffers is based on synchronization with Starbase calls, not on synchronization with Xlib calls. The Starbase graphics library supports double buffering by automatically making a

double_buffer call when an X11 window is opened. It does so with the following parameters of the double_buffer procedure:

MODE	TRUE
DFRONT	TRUE
Number of planes	Same as X11 server double-buffer mode

One example is where an eight-plane display is operating in 4/4 double-buffer mode. When an X11 window is opened, the Starbase double_buffer call is automatically made by Starbase to initiate Starbase double-buffer operation in that window. If you then call the Starbase procedure inquire_fb_configuration, you'll be informed that you have a four-plane device, not an eight-plane device.

Display-Control Policy and Double Buffering

In order for double buffering to appear visually correct within a window, the window must be the current display-control focus window. This allows the Starbase driver to change the display-enable register when switching from one buffer to the other for this window (remember that the display-enable register is a display-control value). Rendering still occurs when the window is not the display-control focus window. However, Starbase double-buffer switches in the non-focus window do not affect which buffer is displayed (the double-buffer switches do affect which buffer is written to). Because X11 renders to both buffers simultaneously, the window borders will always appear correct, regardless of the contents of the window. Likewise, windows containing only Xlib rendering will always appear correct.

Applications That Do Not Use Double Buffering

It is recommended that your Starbase application use inquire_display_mode to determine the double-buffer startup mode of the X11 server and respond accordingly with Starbase double-buffer calls. If the X11 server is operating in double-buffer mode but your Starbase application does not use double buffering (that is, does not use dbuffer_switch and double_buffer), your single-buffered Starbase application only writes to the lower buffer. If another Starbase application operating in the display control focus window calls dbuffer_switch, which switches the displayed buffer to the upper buffer, your single-buffered image will "disappear" until the lower buffer is display-enabled again. When a window

running a single-buffered Starbase program becomes the display-control focus window, the lower buffer is immediately displayed and the window appears correct (and the display stops toggling between the upper and lower buffers).

Summary

The steps to use double buffering are:

1. If you have control of the X11 server double-buffer mode, you know it meets the double-buffer needs of your program. However, if your application program is intended to operate with each of the four X11 server modes on the various displays, you should use `inquire_display_mode` to determine the X11 server double-buffer mode when the window is opened and then respond accordingly with Starbase double-buffer calls.
2. When moving a raw mode Starbase program to X11, you need to be aware of the double-buffer mode it requests. You can use `inquire_display_mode` to determine the X11 server double-buffer mode and then make the appropriate Starbase double-buffer calls.
3. Backing store for obscured windows is supported in double-buffer mode. Refer to the next section for a description of how backing store operates in double-buffer mode.

Backing Store Operation

This section describes backing store support provided by X11. Two terms which essentially mean the same thing are:

- backing store (Xlib terminology)
- retained raster (HP Windows/9000 terminology)

In our discussions, the term **backing store** is used. Backing store is memory used to retain graphics data rendered to obscured portions of a window. Rather than "lose" the graphics information, it is stored in memory. When the obscured portion of the window is exposed, the retained graphics data is transferred from memory to the display frame buffer. A window that has backing store associated with it is referred to as **retained.**

Backing store memory can be either virtual memory or display offscreen memory. The X11 server first attempts to allocate backing store memory for a window in offscreen memory. However, depending on the size of offscreen memory (which varies from display to display) and other usages of offscreen memory (for example, storage of fonts), there may not be enough offscreen memory to support backing store. In this case, the X11 server uses virtual memory.

The advantage of offscreen backing store memory is that rendering is faster. Because the functionality is the same for virtual and off-screen backing store, programs should not need to know where the backing store memory is located; therefore, no mechanism is available to programmatically determine the location of backing store memory.

Use of backing store is optional. When it is not used, graphics operations intended for obscured portions of the window are lost. You can use the X11 window exposure event `XExposeEvent` to re-generate your image. Window exposure events are described in the manual *Programming With Xlib, Version 11.*

Backing Store Cases

There are three cases involving obscured windows and backing store that should be considered:

- Combined mode: In the case of overlay plane windows covering up image plane windows, backing store is not required. The overlay planes reside "on top" of the image planes in physically separate frame buffer planes,

so the graphics rendered to an image plane window which is obscured by an overlay plane window is not lost. However, combined mode backing store is needed to save the contents of an obscured window when one of the following occurs:

- An overlay-plane window obscures another overlay-plane window.
- An image-plane window obscures another image-plane window.
- An image-plane window obscures an overlay-plane window.

- Saving the Contents Beneath a Window: This capability is not automatically supported by the X11 server. In order to save the contents of the windows beneath a new window, the windows being obscured must be retained windows.
- Saving and rendering to an obscured portion of a window: This is the main focus of this section and is discussed in detail below.

Creating an X11 Window Which Supports Backing Store

When an X11 window is created, you can request the window to support backing store. X11 windows which support backing store can be created using either the `xwcreate(1)` command or the `XCreateWindow` procedure, as follows:

Using xwcreate(1)

The `-r` parameter of `xwcreate` is used to specify that a window supports backing store. The following example creates a 300×300-pixel window which supports backing store:

```
xwcreate -r -geometry300x300+10+10 -depth 8  GraphWin
```

Using XCreateWindow

When an X11 window is created with the `XCreateWindow` procedure, it can be created as a retained window using the `backing_store` parameter of the `attributes` structure. `backing_store` can have these three values:

`NotUseful`	No backing store.
`WhenMapped`	Backing store provided only when the window is mapped to the screen.
`Always`	Backing store provided as long as the window exists.

Enabling Backing Store after Window Creation

The `backing_store` parameter in the `XSetWindowAttributes` structure can be changed after the window is created using the `XChangeWindowAttributes` procedure, which will enable backing store in a window which was created without it.

Enabling Starbase Backing Store

When a window which supports backing store is then opened for Starbase output, the Starbase graphics will not be retained unless you also link the Starbase **byte driver** (`/usr/lib/libddbyte.a`) and/or the Starbase **bit driver** (`/usr/lib/libddbit.a`) as appropriate for the depth of the window's visual. For obscured portions of a window, these Starbase drivers draw to virtual memory instead of the display. When the previously obscured portion of the window becomes unobscured, the information is fetched from memory and written to the display by the X11 server.

The bit driver is used for monochrome displays (for example, the HP 98548 display) while the byte driver is used for color displays. The bit driver provides backing store for monochrome displays if it is linked into your program and the window is retained. Likewise, the byte driver provides backing store for color displays if it is linked into your program and the window is retained.

Backing store memory may be either in virtual memory or display offscreen memory. The bit and byte drivers are required only for virtual memory backing store; the Starbase display drivers do the rendering for offscreen backing store. However, because it is typically not possible to guarantee that display offscreen memory will be used for backing store, the bit and/or byte drivers should always be linked into your application when Starbase backing store support is required.

Note The bit driver is only supported on the HP 9000 Series 300 computers, not on the HP 9000 Series 800 computer. This is because there are no monochrome displays supported on the Series 800 computers.

Intermixed Starbase and Xlib Calls

Backing store is supported for Starbase and Xlib when these calls are intermixed in the same program. Both Starbase and Xlib render to the same backing store memory. However, Starbase backing store operation is not affected by the Xlib attributes `backing_planes` and `backing_pixel`. The planes used by Starbase for backing store are controlled by the Starbase procedure `write_enable`. Xlib backing store operation, however, is affected by the `backing_planes` and `backing_pixel` attributes.

Backing Store Control

The following table shows how the `backing_store` attribute and linkage of the Starbase bit and/or byte drivers control backing store operation:

Table 4-8. Control of Backing Store

`backing_store` Value	Bit/Byte Driver Linked	Bit/Byte Drivers Not Linked
`NotUseful`	Neither Xlib nor Starbase retained	Neither Xlib nor Starbase retained
`WhenMapped`	Xlib and Starbase retained when window is mapped	Xlib retained when window is mapped, Starbase not retained
`Always`	Xlib and Starbase always retained	Xlib always retained, Starbase not retained

Depth of Backing Store

Hewlett-Packard displays have various depths, from one-plane monochrome displays to 24-plane full-color displays. For Starbase, the fact that backing store is supported by the Starbase bit driver and byte driver indicates the depth of Starbase backing store. Single-plane backing store is provided for monochrome displays by the bit driver and eight-plane backing store is provided for systems with eight planes or less by the byte driver.

When the number of planes being used by Starbase exceeds eight planes, Starbase backing store is not supported. Thus, for example, when you are using the

HP 98721A in 8/8 double-buffer mode, 12/12 double-buffer mode or 24-plane mode, backing store is unavailable. When you set the `backing_store` attribute to `Always` or `WhenMapped` and the byte driver is linked in, you won't get an error, but nothing will be rendered into backing store memory.

Xlib backing store, however, can exceed eight planes. When the `backing_store` attribute is specified, Xlib backing store is supported for 8/8, 12/12 and 24-plane systems.

Backing Store Operation With Graphics Accelerators

Backing store for the HP 98721A accelerator is not supported because Starbase does not support rendering with the HP 98721A accelerator to an X11 window.

When you use the optional HP 98732A and HP 98556A accelerators, the images in the frame buffer are generated by the accelerator hardware, not by software, and there is no way for the accelerator output to be "fetched" for retention in backing store memory when the window is obscured. Therefore, when a window being rendered into by the HP 98732A or HP 98556A accelerators is obscured, any graphics generated by the accelerator that is intended for the obscured portion is lost.

Users of HP Windows/9000 may recall that the HP 98550 driver (if linked) will automatically be used to render to an obscured window in lieu of the HP 98556 driver. This capability is not supported in an X11 window.

Backing store is supported with the HP 98721A and HP 98732A displays when you use the non-accelerator drivers (that is, the HP 98720 and HP 98730 drivers). However, these drivers provide less functionality than the accelerated drivers. Also, as discussed above, Starbase backing store is only supported for eight planes or less. With the HP 98732A driver, backing store support is only provided for retention of graphics data, not Z-buffer data.

Multiple Starbase Opens

When an X11 window which supports backing store on a color monitor is opened by two Starbase programs, the first program linked with a byte driver and the second program not linked with a byte driver, only Starbase graphics rendered by the first program are retained.

Summary

The main points of backing store operation in double-buffer mode are:

- Because Starbase backing store is supported by the byte driver, a maximum of eight total planes can be retained in backing store. Thus, in double-buffer mode, 4/4 is retainable but 8/8 is not retainable (due to having greater than eight total planes).
- Xlib backing store supports any number of planes up to 12/12 in double-buffer mode.

Window Re-Sizing

When a window which supports backing store is re-sized (either decreased or increased in size), all backing store information is lost. After the resizing occurs, rendering to backing store memory resumes.

Summary of Steps

The steps to use backing store are:

1. Create a retained X11 window as described previously using either the `xwcreate(1)` command or the `XCreateWindow` procedure. Alternately, the `XChangeWindowAttribute` procedure can be used to enable backing store after the window is created. All these methods ensure that Xlib rendering will be retained.
2. To enable Starbase backing store, link the bit driver and/or byte driver into your application.
3. Open the window with a Starbase driver that supports backing store. The HP 98732A and HP 98556A accelerator drivers are examples of drivers which do not support backing store.

Window Resizing Operation

Starbase Window Size

An X11 window of any size up to 32K×32K pixels can be created. When this window is opened by Starbase with a `gopen` call, the Starbase library automatically learns the window size. You can then use the Starbase procedure `inquire_sizes` to determine the size (in pixels) of the window.

Effects of Re-sizing the Window

If you change the size of the window after it is opened by Starbase, rendering will be in the same position relative to the upper left corner of the window. However, the Starbase graphics library is not aware that the window has changed size; the Starbase procedure `inquire_sizes` will return the old (prior to re-sizing) window dimensions. Because the Starbase library is not aware of window re-sizing, the scaling of Starbase graphics is not changed. For example, if your 400×300 pixel X11 window is opened by Starbase and then you later reduce the window size to 200×200 pixels, Starbase will not reduce the scaling of its graphics. Starbase does, however, clip output to the window's new size when rendering extends beyond the it.

If you want to change the scaling of your Starbase graphics, you can detect a resize request with the X11 event `ConfigureNotify`; window size information is provided with the event. The subsequent steps that a program should take depend on whether the window has been increased or decreased in size, as follows:

- If the window size is decreased in size, you can call the Starbase procedure `set_p1_p2` to specify a correspondingly smaller drawing area. If you want to change the size of the region of world coordinate space that is visible in the window, you can use either the `vdc_extent` or `viewport` procedure. The fact that the Starbase graphics library does not know that the window is smaller is not a problem because the `set_p1_p2` procedure limits Starbase output to the smaller boundaries marked by Process 1 and Process 2. This re-scaling does not apply to Starbase device coordinate graphics. If your program is using device coordinate procedures, you may want to adjust any program scaling based on the re-sized window.

- If the window size is increased, it is not possible to use the Starbase procedure `set_p1_p2` to enlarge the drawing area to encompass the larger window. Because the Starbase library is not aware of the increase in window size, it does not allow the `set_p1_p2` procedure to be used to increase the drawing area beyond the original size of the window.

In order for the Starbase library to be aware of a change in window size, it is necessary to close the window with the `gclose` procedure and then re-open the window with the gopen procedure. After the window is opened, Starbase will be aware of the new size and `inquire_sizes` can be used to obtain the size.

Affect on Xlib

Because Xlib graphics primitives are in device coordinates (that is, pixels), there are no Xlib scale factors which require changing if the window size is changed. If the window is increased in size, the program can use the larger area without any additional steps. If your application provides its own scaling (for example, does 2D transformations), you can detect the X11 event `ConfigureNotify` and change your program's scale factor accordingly.

X11 Cursor and Starbase Echo Operation

Echoes and cursors refer to graphical entities on the screen which track the motion of some type of input device (for example, the server's pointer device). Several terms which are often used interchangeably are:

- Cursor
- Echo
- Sprite
- Pointer

In this manual, only the terms cursor and echo are used. These terms are defined as follows:

- **Cursor:** This is the graphics entity which follows the X11 pointer device. This is referred to as the **X11 cursor.**
- **Echo:** This is the graphics entity controlled by Starbase, also called the **Starbase echo.** The Starbase echo can be "attached" to an input device (for example, an HP-HIL tablet). As the input device moves, the Starbase echo tracks this motion. Starbase also provides the capability to position the echo programmatically.

Starbase Raster and Vector Echoes

Starbase supports both raster echoes and vector echoes. A vector echo is composed of one or more vectors. An example of a vector echo is a full-screen cross hair. A raster echo is composed of a user defined bit pattern which is specified by the Starbase procedure `define_raster_echo`. For more information on Starbase echoes, refer to *Starbase Graphics Techniques* and to the specific Starbase driver documentation in the *Starbase Device Drivers Library Manual.* The Appendix in the *Starbase Device Drivers Library Manual* describes several `gescape` opcodes that provide additional control of Starbase echoes.

The type of echo chosen affects whether the echo is placed in the overlay planes or the image planes on certain displays. Vector cursors are best suited for the overlay planes because they are typically monochromatic (the default is white) and can live with the limited colors available in the overlay planes. Raster cursors typically reside in the image planes, permitting them to use the colors that are available in the image plane's color map.

Hardware Support for Cursors and Echoes

Certain displays provide hardware support for cursors and echoes. For example, the HP 98720 display reserves one of its four overlay planes as a cursor plane. Likewise, the HP 98730 display provides a cursor plane when only three of the four overlay planes are being used by the X11 server. In addition, the HP 98730 provides a hardware cursor. Refer to the *Starbase Device Drivers Library Manual* for a description of the cursor support provided by each display.

Picking up the Cursor or Echo

Whenever Xlib or Starbase rendering occurs in a window which contains an X11 cursor or a Starbase echo, the cursor or echo is momentarily "picked up". This means that the echo or cursor is removed from the screen and the graphics data that was previously there is restored. This allows rendering to occur with the correct data in the frame buffer. Once the rendering is done, the cursor or echo is moved back on the screen after a copy of the area "under" the cursor is saved (this saving only occurs for raster echoes).

Starbase vector echoes in the cursor planes do not need to be picked up and restored when rendering is performed in the image planes. This is because the cursor planes reside in physically separate planes "on top" of the image planes.

Raw Mode Starbase Echoes

Because the image planes can be opened in raw mode by Starbase when the X11 server is operating in overlay mode, it is possible to have a raw-mode Starbase echo operating in the image planes at the same time Starbase echoes are operating in overlay plane windows. Because opening of the overlay planes is not supported when the X11 server is operating in Image Mode, it is not possible to have overlay-plane Starbase echoes while X is running.

X11 Cursors and Starbase Echoes

The following table shows default positions where the Starbase echo and X11 cursor reside for each of the X11 server operating modes. The Starbase gescape `R_OVERLAY_ECHO` can be used to change the default echo placement from the image planes to the overlay planes. "Shares" means that the echo or cursor is rendered into the same planes used by Starbase or Xlib for rendering.

Table 4-9. Default Starbase Echo and X11 Cursor Placement in the Four X11 Server Modes

	Overlay Mode	Image Mode	Stacked Screen Mode	Combined Mode
300 Medium and High Res Displays		Echo and cursor share the image planes		
98548A, 98549A		Echo and cursor share the image planes		
98550	Echo placed in whatever planes (image or overlay) are opened. Cursor always in overlay planes.	Echo and cursor share image planes.	Echo and cursor share overlay planes if overlay plane X11 window opened. Echo and cursor share image planes if image-plane window opened.	

Table 4-9. Default Starbase Echo and X11 Cursor Placement in the Four X11 Server Modes Continued

	Overlay Mode	Image Mode	Stacked Screen Mode	Combined Mode
HP 98721	Echo shares 3 overlay planes if overlay plane X11 window opened. Raster echo in image planes if image planes opened. Vector echo in cursor plane if image planes opened in raw mode. Cursor shares 3 overlay planes.	Raster echo in image planes if image plane X11 window opened. Vector echo in cursor plane if image plane X11 window opened. Cursor shares image planes.	Echo shares 3 overlay planes if overlay plane X11 window opened. Raster echo in image planes if image plane X11 window opened. Vector cursor in cursor plane if image plane X11 window opened. Cursor shares image planes for image plane window, shares overlay plane for overlay plane window.	

Table 4-9. Default Starbase Echo and X11 Cursor Placement in the Four X11 Server Modes Continued

	Overlay Mode	Image Mode	Stacked Screen Mode	Combined Mode
HP 98731	If X11 server is in 3 overlay planes and Starbase image planes are opened, raster and vector echoes are in cursor plane. If X11 server is in 4 overlay planes and Starbase image planes opened, no Starbase echoes are supported. X11 cursor uses hardware cursor. If an overlay plane window (3- or 4-plane) window is opened, Starbase echoes share the overlay planes.	Starbase echoes from a gopen of an image-plane window placed in the cursor plane. X11 cursor uses hardware cursor.	If the X11 server is in 3 overlay planes and an image-plane window is opened, Starbase echoes are placed in the cursor plane. If the X11 server is in 4 overlay planes and an image-plane window is opened, Starbase echoes are not supported. X11 cursor uses hardware cursor. If an overlay-plane window (3- or 4-plane) window is opened, Starbase echoes share the overlay planes.	If the overlay plane window is 3 planes and the Starbase echo is a vector cursor, it is placed in the cursor plane. If the overlay plane window is 4 planes, all Starbase echoes are in the overlay planes. If an overlay plane window (3- or 4-plane) is opened, Starbase echoes share the overlay planes. X11 cursor uses hardware cursor.

Starbase Tracking in an X11 Window

Starbase echo tracking is permitted from any Starbase input device (including an X11 window) to any Starbase output device (again, including an X11 window). Listed below are several examples of Starbase tracking:

1. Same Window Used for Input and Output: This case is activated by doing a Starbase **gopen** of the X11 window as both an input device and an output device (with the **gopen** ⟨*kind*⟩ parameter set to `OUTINDEV`). In this case, the Starbase echo tracks the position of the X11 cursor. Because the X11 cursor and Starbase echo both point to the same spot, the X11 cursor is removed while asynchronous tracking is enabled in the same window that contains the X11 cursor. When tracking is disabled, the X11 cursor returns. If the pointer device is used in an attempt to move the Starbase echo outside of the window, the echo is "pegged" against the inside border of the window and the X11 cursor re-appears outside of the window.

2. Different Input and Output Windows: It is possible to open one X11 window as the input window and open another as the output window, then enable tracking between them. When the X11 cursor is moved in the input window, the Starbase echo tracks its position in the output window. Tracking is proportionally correct when the windows are different sizes.

3. Non-Window Related Input Device: In this case, input is obtained from an input device not associated with a window (for example, a tablet). The X11 cursor is not affected (that is, it keeps tracking the window pointer device) and the Starbase echo tracks the tablet. The X11 cursor and the Starbase echo can be active in the same window.

HP 98732A Hardware Cursor

The HP 98732A 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.

HP 98556A 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.

X11 and Starbase Synchronization

Both Xlib graphics and Starbase graphics are buffered to improve their performance. These buffering schemes are implemented in separate processes and are completely independent. When your application is rendering both Xlib and Starbase graphics to the same window and when the order in which the graphics primitives are rendered is important, you should synchronize your program.

`XSync` is used to flush and wait for all Xlib graphics primitives to be rendered to the window by the X11 server. The procedure `make_picture_current` is used to flush the Starbase output buffer. The outline of a program mixing Xlib and Starbase graphics calls, and providing synchronization follows:

```
X initialization
Starbase initialization
    :
X graphics primitives
Xsync( ... )
    :
Starbase graphics primitives
make_picture_current( ... )
    :
X graphics primitives
    :
X graphics primitives
Xsync( ... )
    :
Starbase graphics primitives
    :
Starbase graphics primitives
make_picture_current( ... )
    :
```

5

Raster Text Operation

Introduction

This chapter assumes that you are already familiar with the programmatic interface for the particular font library that you plan to use. This combined with the information in this chapter will enable you to successfully develop your raster text application.

Raster text is represented by patterns of individual pixels which form a character on the display. **Stroke text** is represented by a combination of short vectors which form the character. This chapter describes only raster text. The stroke text provided by Starbase is discussed in *Starbase Graphics Techniques.*

Font Formats and Character Sets

Generation of raster text requires accessing files on disc which contain the actual raster patterns; these are called **font files.** In discussing font files, we need to distinguish between font file format and font character set, as follows:

The **font file format** (or font format) is the disc format the font is stored in. The font format specifies the organization of the raster data in the file. There are several different font formats. For example, the font format associated with the 6.2 release of HP Windows/9000 is different than the font format associated with the 6.5 release of HP Windows/9000.

The **font character set** (sometimes called the "character set" or just "font") specifies the raster pattern of characters that are displayed on the screen. The character sets provided with HP Windows/9000 are referred to as the Fast Alpha/Font Manager (FA/FM) fonts. In some cases, the same character set is available in different font file formats. For example, both X11 and

HP Windows/9000 provide the Roman8 character set but in different font file formats. Character sets are further distinguished by:

- The *number* (or index) that is used to access each character. For example, the character "A" is character number 65 in most character sets.
- The *size of the index.* Seven bits (specifying up to 128 characters), eight bits (specifying up to 256 characters) and 16 bits (specifying up to 64K characters).
- The character set *cell size* specifies the width and height of the space occupied by each character (for example, 8 pixels by 16 pixels).
- The character *style* specifies different ways of presenting a character set (for example, bold or italics).

8-Bit Fonts versus 16-Bit Fonts

An 8-bit font means that the characters can be indexed by an 8-bit value. This limits 8-bit fonts to a maximum of 256 characters. A 16-bit font is indexed by a 16-bit value and thus can represent up to 64K characters. 16-bit fonts are typically used for Far East fonts such as Kanji.

To display a particular 8-bit character string, you pass the character string (for example, "ABC") into the font library procedure. With 16-bit fonts (supporting up to 64K characters), you have to provide an array of 16-bit indices to the font procedures which specifies the characters to display instead of typing characters into the font procedure's character string.

HP-15

HP-15 is an encoding scheme developed by Hewlett-Packard that is often used to represent Asian fonts. For example, there is an HP-15 Kanji font which specifies the encoding of several thousand raster patterns representing the Kanji characters. HP-15 supports mixed 8- and 16-bit fonts. This means that you can provide an array of numeric indices which reference both 8- and 16-bit fonts. The font procedure sorts out whether an 8- or 16-bit character is being accessed by each index. This is done as follows: The first byte of each 16-bit index is examined by the font procedure to determine the range of the byte. A certain range indicates that the byte represents a single character (an 8-bit font). Another range indicates that the byte represents the first byte of a two byte index. In this

case, the font procedures examine both bytes to determine the two-byte (16-bit) index. In this manner, you can specify both 8-bit fonts and 16-bit fonts in one array of indices. HP-15 fonts such as Kanji are shipped in various file formats, depending on the intended window environment.

Summary of Changes

The major changes to text operation that have occurred with the 6.5 (Series 300) and 3.1 (Series 800) releases of HP-UX are:

- The Fast Alpha/Font Manager (FA/FM) libraries can be used in an X11 window. Prior to the 3.1 (Series 800) and 6.5 (Series 300) HP-UX releases, these libraries were only usable in raw mode and with HP Windows/9000 (Series 300 only). As with raw mode or HP Windows/9000 operation, the file descriptor returned by the Starbase open of the X11 window is used by the FA/FM routines.
- The FA/FM libraries and the X11 raster font procedures now share a common font file format. This format is based on the X11 format, not on the FA/FM font file format. This permits any FA/FM font or X11 font to be used by either the FA/FM libraries or the X11 font library.

Raster Text Capabilities

In describing the raster text capabilities in each of the window environments, we need to discuss the different:

- Font libraries
- Character sets (fonts)
- Font file formats

Font Libraries

The three font libraries are:

- Fast Alpha/Font Manager (FA/FM) Libraries: These libraries were originally developed as part of the HP Windows/9000 product but can also be used by raw mode Starbase.
- X10 font library: Refer to *X10 Xlib Programming Manual* for a description of the X10 font library.
- X11 font library: Refer to *Programming With Xlib, Version 11* for a description of the X11 font library. The X11 font library is the same as the X11 revision A.00 font library.

Character Sets (Fonts)

The three major groups of fonts available are listed below. These font groups should not be confused with the various font file formats that the fonts are provided in. For example, the FA/FM fonts are available in two file formats, a pre-3.1/6.5 format and a format based on the X11 format.

- FA/FM fonts: FA/FM fonts are encoded as 8-bit fonts and HP-15 fonts. HP-15 is used to encode the Kanji font.
- X10 fonts: These fonts were released by MIT with X10 and are 8-bit fonts.
- X11 fonts: These fonts were released by MIT with X11 and contain both 8-bit fonts and 16-bit fonts.

Font File Formats

The font file formats that are supported on disc are:

- Fast Alpha/Font Manager (FA/FM) font file format: The FA/FM font file format supports 8-bit fonts and HP-15 fonts.
- X10 font file format: X10, as released by MIT, only supported 8-bit font files. HP has extended the X10 server to support loading of HP-15 bit font files in X10 format.
- X11 Server Natural Format (SNF) font files: X11, as released by MIT, supported 8-bit and 16-bit SNF fonts. HP has provided an SNF HP-15 font. SNF, however, has system dependencies (for example, the byte padding at the end of records can vary), so vendors have developed fonts with different Server Natural Formats.
- Bitmap Distribution Format (BDF), developed by Adobe Systems, Inc.: While SNF may change from one implementation to another, BDF is expected to remain unchanged. A font compiler is provided with X11 to compile BDF fonts into SNF fonts; refer to *Programming With Xlib, Version 11* for more details.

The table below summarizes the font libraries, font character sets and font file formats which are supported in the different window systems. Raw mode and HP Windows/9000 are grouped together because the font capabilities are identical.

Table 5-1. Font Libraries, Font Character Sets and Font Formats

Window System	Supported Font Library	Supported Fonts	Font File Format
Pre-3.1/6.5 Raw Mode and Windows/9000	FA/FM Library	8-Bit: FA/FM fonts; HP-15: FA/FM fonts	FA/FM format
X10	X10 font procedures	8-Bit: X10 fonts; HP-15: HP X10 Kanji	X10 format
X11 revision A.00	Xll font procedures	8-bit: X11 fonts; 16-bit: X11 Kanji; HP-15: HP X11 Kanji	Server Natural Format (SNF), BDF
X11	X11 Xlib font procedures, FA/FM Library	8-bit: X11 fonts, FA/FM fonts; 16-bit: X11 Kanji; HP-15: HP X11 Kanji	SNF, BDF
3.1/6.5 Raw Mode	FA/FM Library	8-bit: X11 fonts, FA/FM fonts; HP-15: HP X11 Kanji	SNF, BDF
3.1/6.5\HP Windows/9000	FA/FM Library	8-bit: FA/FM fonts; HP-15: HP X11 Kanji	SNF, BDF

Main Points From the Previous Table

- Only the X11 font procedures can be used in an X11 revision A.00 window. Both the X11 font procedures and the FA/FM font procedures can be used in an X11 window.
- In 3.1/6.5 raw mode, the FA/FM library can access either the FA/FM fonts or the X11 fonts. However, in 3.1/6.5 HP Windows/9000, only the FA/FM and HP-15 fonts can be used. Use of the X11 fonts is not supported.

Fonts Used by the FA/FM Library

The 3.1/6.5 FA/FM font loader (which loads fonts from disc into virtual memory) has been modified to only load SNF fonts, not fonts in the pre-3.1/6.5 FA/FM font format. Therefore, the FA/FM fonts have been converted to SNF and provided with the 3.1/6.5 HP-UX releases. Several points should be noted about this:

- The functionality of the FA/FM libraries is unchanged, only the font file format is changed. Access to X11 fonts by the FA/FM libraries is only supported in raw mode. HP Windows/9000 does not support use of X11 fonts.
- The pre-3.1/6.5 FA/FM font files cannot be loaded by a later FA/FM font loader. To support pre-3.1/6.5 executables which reference FA/FM fonts, the FA/FM font files are shipped with the 3.1/6.5 releases of HP-UX.
- When you compile a pre-3.1/6.5 FA/FM program (which references the pre-3.1/6.5 FA/FM font files by name) and then link the program with the later FA/FM libraries (which only know how to load SNF fonts) the following support is provided to avoid changing the font file names in your source code.

  When the referenced FA/FM font file is not an SNF font, the FA/FM font loader looks for a related SNF font in an associated subdirectory. For example, if your pre-3.1/6.5 program accesses the FA/FM font `/usr/lib/raster/8x16/lp.8U`, this font cannot be loaded by the 3.1/6.5 FA/FM font loader because it is not an SNF font. Therefore, the font loader looks for the SNF font `/usr/lib/raster/8x16/SNF/lp.8U.snf` and loads it.

The following diagram shows several examples of how the FA/FM SNF and non-SNF fonts are related:

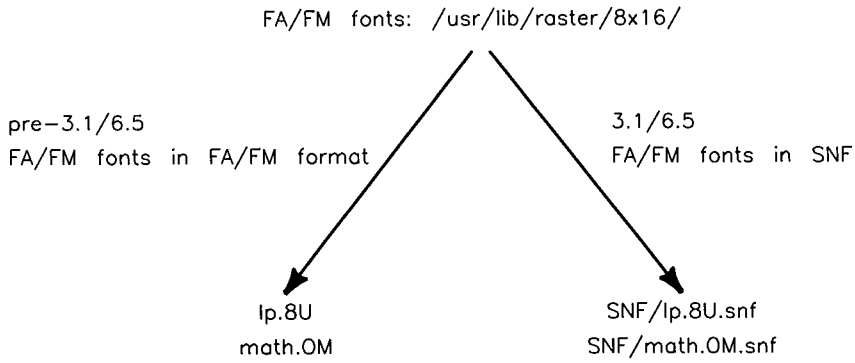


Figure 5-1. FA/FM Fonts in FA/FM Format and in SNF

The following table summarizes what font is used based on the nature of your program:

Table 5-2. FA/FM Font File Usage

Application	Font File Used	Example Font File
Pre-3.1/6.5 executable	Pre-3.1/6.5 FA/FM fonts in FA/FM format	`usr/lib/raster/8x16/lp.8U`
Pre-3.1/6.5 source which references pre-3.1/6.5 FA/FM fonts but is linked with later libraries	FA/FM font files in SNF	`usr/lib/raster/8x16/SNF/-lp.8U.snf`
FA/FM program which references 3.1/6.5 FA/FM fonts	FA/FM font files in SNF	`usr/lib/raster/8x16/SNF/-lp.8U.snf`

6

Input Operation

Introduction

This chapter assumes that you are already familiar with how Starbase input works in a raw environment; if not, refer to the "Starbase Input" chapter in *Starbase Graphics Techniques*. This chapter also assumes that you are familiar with Xlib input operation and the X11 input extensions. For more information on Xlib input operation, refer to *Programming With Xlib, Version 11*. Only input from HP-HIL devices is covered in this chapter. Programmers may also choose to obtain input from non-HIL devices, such as RS-232 terminals or HP-IB plotters. While the HP X11 environment supports the sharing of HIL devices by different programs, it does not support the sharing of non-HIL devices.

The main topics covered in this chapter are:

- Input device sharing.
- Input focus policy.
- Starbase input in a raw, HP Windows/9000, X10, and X11 (revision A.00) environment.
- Starbase input in an X11 environment.
- Examples to aid in moving a Starbase input application to the X11 environment.

Overview of Input Operation

Starbase input in an X11 window provides full-functionality Starbase input through a window. Input devices can be shared with other programs with the X11 server arbitrating which program receives input at any given moment. It is possible for programs to work in an X11 environment without any code changes; when code changes are required, the reasons will probably be:

- The program has hard-coded values for the parameters of the Starbase gopen statement(s).
- The program is written in a device-dependent manner that make it impossible to work in a window without code changes.

It is always desirable to write programs in as portable and device-independent a manner as possible. Typically, all that needs to be changed in order for your program to run in an X11 window is the program link sequence and the gopen statement. When the gopen parameters are passed into the program at startup time, no source code changes should be required.

Input Device Sharing

Input device sharing means that multiple programs operating in different windows may obtain input from the same HIL input device. This is beneficial because independently developed programs can access any of the HIL input devices without interfering with other programs which might also be accessing these same devices. For example, with X11, multiple programs operating in different windows may each read the position of the window system's pointer device.

A program permits an input device to be shared with other programs by opening the device in shared access mode. Exclusive access mode means that only one program can open and access an input device at a time. Raw mode Starbase supports only exclusive access to input devices. When a raw mode Starbase program opens a mouse with the Starbase HP-HIL driver, another Starbase program cannot obtain input from the same mouse.

Several Starbase drivers permit programs running in an X11 environment to share the same HIL devices. While this is a new feature with X11, input device sharing has always been supported to some extent by other window systems. For example, programs operating in different X10 windows can each receive input from the X

server's pointer and keyboard. HP Windows/9000 also permits programs to share the window manager's locator and keyboard.

With X11, programs can share the pointer device and keyboard, and the other HIL devices. Input operation has been expanded to include both Xlib and Starbase sharing of the same input devices.

Note Even in an X11 window, the user can still do an exclusive open of an HIL device. However, this is not recommended because it impacts the ability of other programs to share the same workstation resources.

Input Through A Window

A device/window combination means that a program receives input from a combination of a certain device and window. Thus, a program desiring input must specify both an HIL device and a window. The manner in which programs specify the device and window depends on the input library used. Programs using X11 Xlib specify the desired input device and window with the `XSelectInput` procedure. Starbase programs specify the desired input device and window with the `gopen` procedure. The `path` parameter of the `gopen` procedure specifies both the device and window. The file designator returned by the `gopen` procedure is used to receive input from the designated device/window combination.

Input Focus

The capability to direct the input to one window at a time is supported by X11. The window that receives the input from a particular device is said to be "focused" (X11 terminology) or "selected" (HP Windows/9000 terminology). Because of the emphasis placed on X11, the term "focused" is used in this manual. When a window comes into focus, the input stream from the requested device(s) is directed to that window. When a window goes out of focus, the input stream from the requested device(s) no longer goes to that window.

Input Focus Policy

The policy which controls how the input focus changes from one window to another is referred to as the **focus policy.** Different window systems implement different focus policies. For example, HP Windows/9000 implements an explicit window selection policy for keyboard input. The selected window receives all keyboard input regardless of the position of the HP Windows/9000 locator.

The X Window System implements a default focus policy which can be changed by a window manager. The default X11 focus policy for keyboard input is a cursor tracking policy. The window which contains the X11 cursor receives the keyboard's input. Some window managers (for example, `uwm`) do not implement a focus policy which means that the default server policy is used. Other window managers implement an explicit focus policy similar to HP Windows/9000. Still others (for example, `hpwm`) implement the cursor tracking policy by default, while allowing a user to explictly select a window. Refer to the documentation for your window manager to determine what type of focus policy, if any, it implements.

The diagram below should help clarify the concept of input focus. Program 1 has requested an input event through Window 1 from the HIL input device. Program 2 has requested an input event through Window 2 from the same input device. Assume that the focus policy is a cursor tracking policy and that the X11 cursor is in Window 1, causing Window 1 to be focused. In this case, the input event is propagated only through Window 1 to Program 1. Program 2 will have no visibility of the input event.

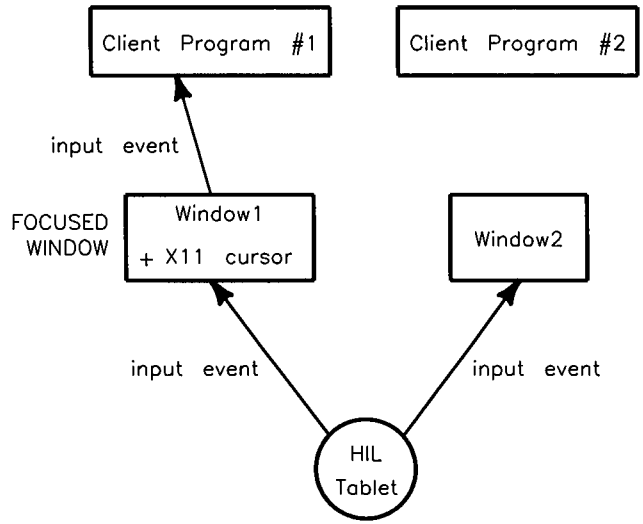


Figure 6–1. Input Focus Rule: Routing of Input Events

Starbase Input in a Raw Environment

In porting an application from raw mode to X11, no window system is running in raw mode, so Xlib input cannot be used. Starbase input procedures and input from the libraries built on top of Starbase are supported. For more information on Starbase raw mode input, refer to *Starbase Graphics Techniques.*

The following diagram shows the data paths for Starbase input in a raw environment:

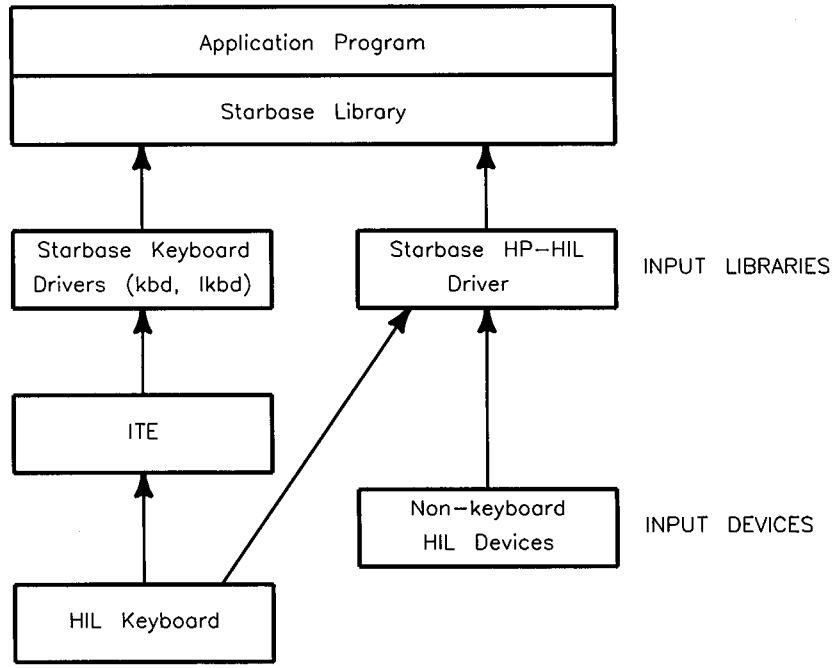


Figure 6-2. Starbase Input in Raw Mode

The main points about Starbase input in a raw environment are:

1. Three Starbase drivers can be used to obtain raw mode input from HIL devices:
    a. Starbase `kbd` driver
    b. Starbase `lkbd` driver
    c. Starbase HP-HIL driver.

2. The Starbase HP-HIL driver can access any HIL device. The Starbase `kbd` and `lkbd` drivers can only access keyboards. The `kbd`, `lkbd` or HIL drivers should not be used to simultaneously access the same keyboard.
3. The `kbd` and `lkbd` drivers receive their information through the ITE (Internal Terminal Emulator) file. See the *HP-UX System Administrator Manual* for more information.

Refer to the *Starbase Device Drivers Library Manual* for more information about these drivers.

Starbase Input in HP Windows/9000

Understanding the Starbase input model for HP Windows/9000 is useful in porting an application from from HP Windows/9000 to X11. Prior to reading this section, you should read "HP Windows/9000 Device Driver" in the *Starbase Device Drivers Library Manual*

The following diagram shows the data paths for Starbase input in an HP Windows/9000 environment:

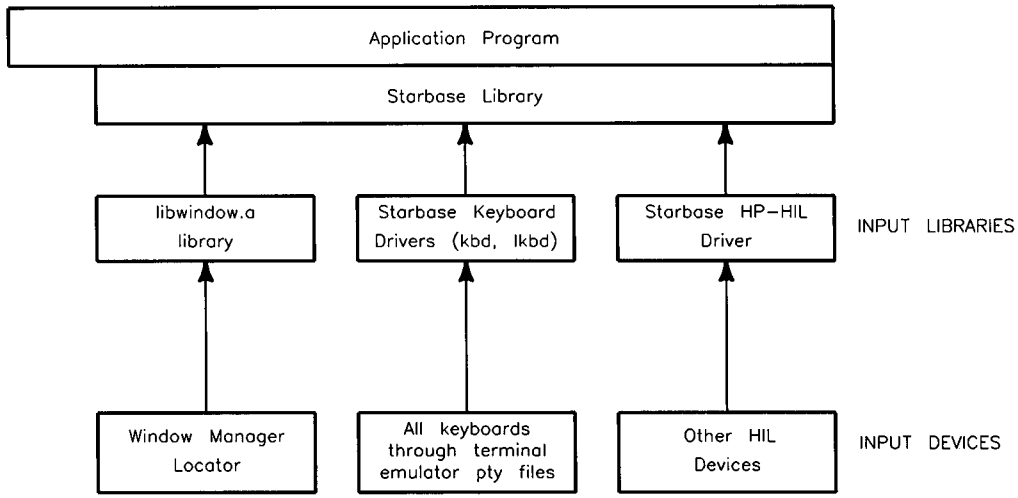


Figure 6-3. Starbase Input in HP Windows/9000

The main points concerning Starbase input in HP Windows/9000 are:

1. The window manager's locator device can be shared by multiple programs. To obtain Starbase input from the window manager's locator device, the program must be linked with the library `libwindow` as shown. The program must also do a Starbase **gopen** of the window using the raw mode display driver (for example, **hp98550**) and specify that the window is either an `INDEV` or `OUTINDEV`.
2. The HP Windows/9000 window manager opens all keyboards on the HIL loop. Therefore, Starbase cannot open these keyboards with the HIL driver.

3. The Starbase `kbd` and `lkbd` drivers receive input via `term0` window `pty` files. This parallels raw mode Starbase where these drivers receive input from `tty` files. Since HP Windows/9000 shares keyboard input among multiple `term0` windows, keyboard input can be shared between multiple Starbase programs. The selected window will receive the keyboard input.
4. Sharing of other HIL devices (for example, a data tablet) is not supported by HP Windows/9000; the HIL devices are available on a first-come, first-served basis only.

Input in an X10 Environment

The diagram below shows the data paths for Xlib and Starbase input in an X10 environment:

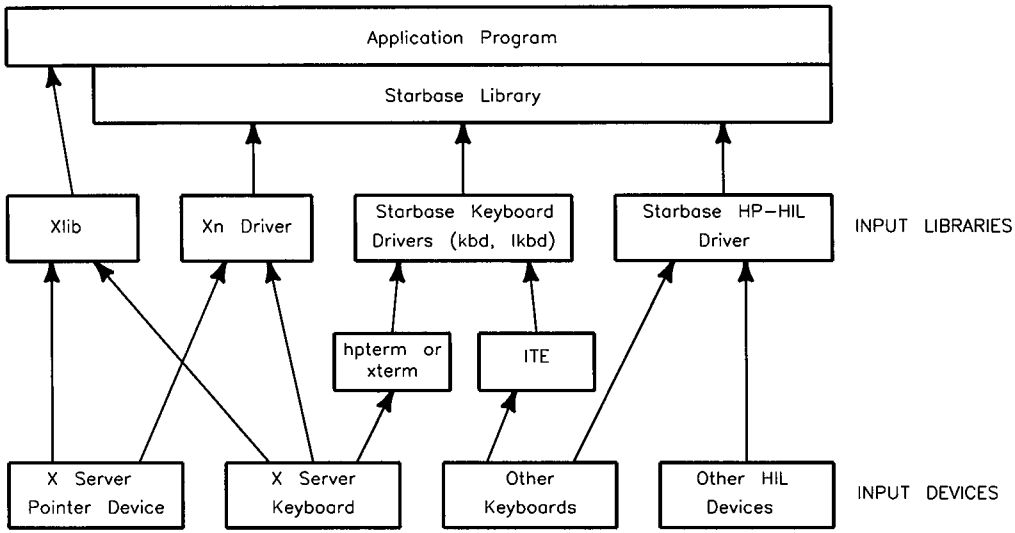


Figure 6-4. Input Operation in an X10 Window

Main Points of the Previous Diagram

1. By default, the HP X10 server opens and uses all input devices on the HIL loop at startup. The server treats every device either as a part of a logical pointer device or a logical keyboard device. The server can be told which devices to use and will still treat each device as a part of a logical pointer or keyboard device, not as separate physical devices. Therefore, the boxes, "Other Keyboards" and "Other HIL Devices" describe HIL devices that the X10 server isn't using.
2. Programs using Xlib can only receive input from the server's logical pointer and keyboard. These devices are shared with other programs.

3. The Starbase Xn device driver communicates with the X server via Xlib and therefore only receives pointer and keyboard input. These devices are shared with other programs.
4. The Starbase keyboard drivers (`kbd` and `lkbd`) can receive input from the X10 server's keyboard via terminal emulator window `pty` files. Since the X server shares keyboard input with all windows, keyboard input can be shared between multiple Starbase programs.
5. The Starbase keyboard drivers can also receive input from keyboards that the X10 server isn't using via the ITE's device file. Refer to the *HP-UX System Administrator Manual* for a description of these device files.
6. The Starbase HP-HIL driver can be used to receive input from other keyboards and other HIL devices. These devices can only be opened for exclusive access.

X11 Revision A.00

This section provides an overview of input in an X11 revision A.00 environment. Understanding this information is useful in porting an application from this environment to the newer X11 environment. The following diagram shows the data paths for Xlib and Starbase input in an X11 (revision A.00) window:

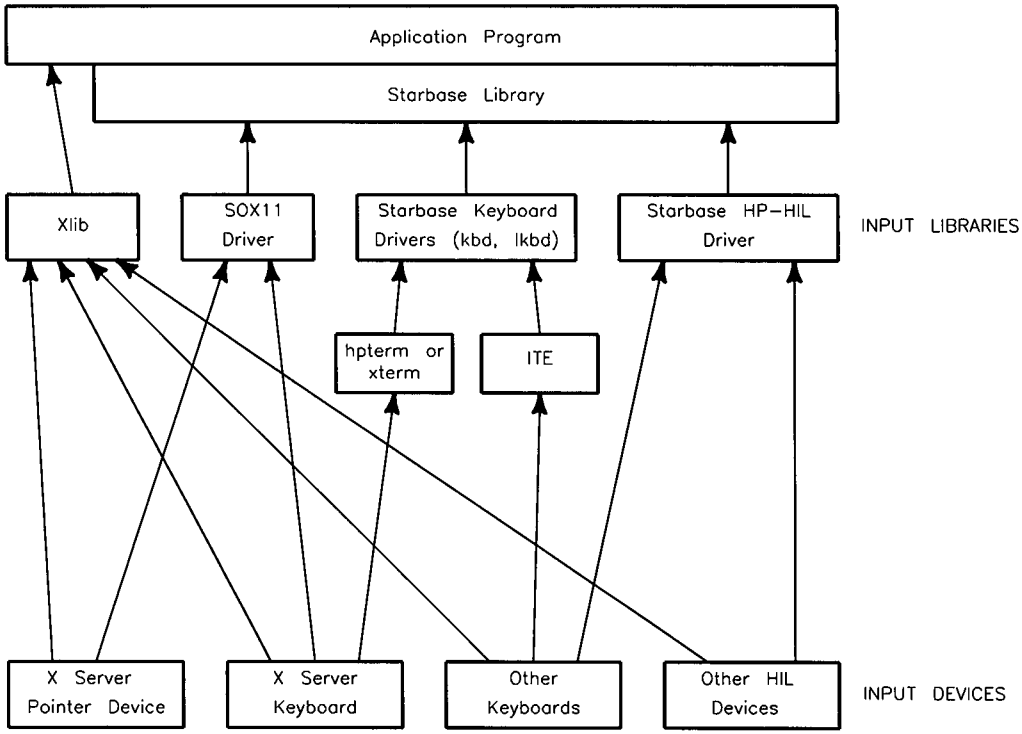


Figure 6-5. Input Operation in an X11 Revision A.00 Window

Main Points of the Previous Diagram

1. The X11 (revision A.00) server, by default, only opens and uses one keyboard and one pointer device. All other devices on the HIL loop are available for use by other applications. The server can be configured to open more than one physical device to use as its logical pointer and

keyboard (this provides backwards compatibility with the HP X10 server). Therefore, the boxes, "Other Keyboards" and "Other HIL Devices" describe HIL devices that the server isn't using as its logical keyboard and logical pointer.

2. Programs using Xlib and the HP Xlib input extension can receive input from all HIL devices. These devices are shared with other programs.
3. The Starbase SOX11 device driver communicates with the server via Xlib but only receives input from the server's pointer and keyboard. These devices are shared with other programs.
4. The Starbase keyboard drivers can receive input from the server's keyboard via terminal emulator window `pty` files. Since the server shares keyboard input with all windows, keyboard input can be shared between multiple Starbase programs.
5. The Starbase keyboard drivers can also receive input from keyboards that the server is not using via the ITE file.
6. The Starbase HP-HIL driver can receive input from other keyboards and other HIL devices. These devices can only be opened for exclusive access; shared mode access is not supported by revision A.00 of X11.
7. Each of the other keyboards and other HIL devices can only be used by the X server (and thus shared by multiple programs) or by one Starbase program (exclusive access) at one time.

X11 Input Operation

There are two ways to receive input in an X11 window:

1. Xlib input procedures: Programs can make X11 Xlib calls to obtain input from HIL devices. Note that X11 input works the same as X11 revision A.00, except that the HP Xlib input extension has been enhanced to support input features required for Starbase input within X11. X11 input operation is described in the manual *Programming With Xlib, Version 11.*
2. Starbase input procedures: Starbase input procedures can be used to obtain input from HIL devices. There are five input libraries and drivers which provide Starbase input through an X11 window.

Input Data Paths

The following diagram shows the data paths for Xlib and Starbase input in an X11 environment. The HIL input devices are grouped into the following four categories:

- *X Server Pointer Device:* This is the pointer device used by the X11 server (typically a mouse). When the server is configured to use more than one physical device as one logical pointer device, this box represents all of those devices.
- *X Server Keyboard:* This is the keyboard used by the X11 server. When the server is configured to use more than one physical keyboard as one logical keyboard device, this box represents all of these keyboards.
- *Other Keyboards:* This represents all keyboards other than the X Server Keyboard.
- *Other HIL Devices:* This represents all HIL devices other than the above three categories.

The arrows from the "Input Devices" to the "Input Libraries" indicate the paths that the input libraries support.

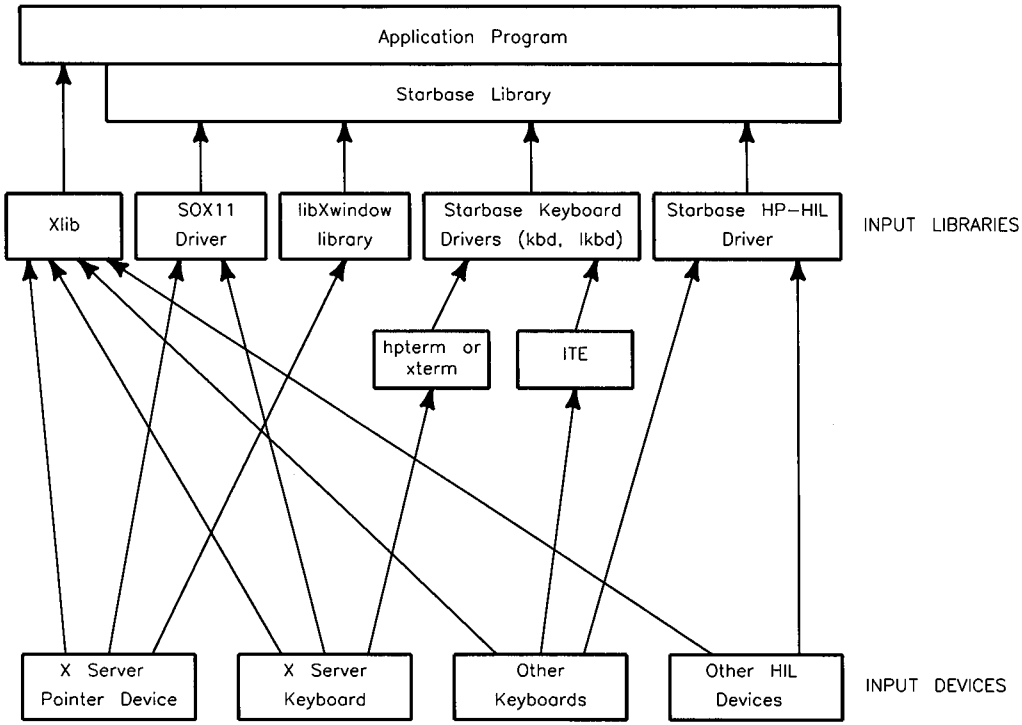


Figure 6-6. Input Operation with X11

Main Points of the Previous Diagram

1. Xlib input
   - Programs using Xlib and the HP Xlib input extension can receive input from all HIL devices. These devices are shared with other programs. Xlib input is described in *Programming With Xlib, Version 11.*
2. SOX11 input
   - The Starbase SOX11 device driver receives input from the server's pointer and keyboard. These devices are shared with other programs.
3. libXwindow library input

- The Starbase libXwindow library receives input from the X server's pointer device and always opens the pointer for shared access.

4. kbd, lkbd input

   - These Starbase keyboard drivers can receive input from the X11 server's keyboard via terminal emulator window pty files. Since the X11 server shares keyboard input with all windows, keyboard input can be shared between multiple Starbase programs.
   - The Starbase keyboard drivers can also receive input from keyboards that the X11 server isn't using via the ITE file.

5. HP-HIL input

   - The Starbase HP-HIL driver has been enhanced for the 3.1/6.5 release of HP-UX to support input device sharing. With raw mode Starbase input and Starbase input prior to the 3.1/6.5 release, the HP-HIL driver supported only exclusive access to HIL devices.
   - Each of the "Other Keyboards" and "Other HIL Devices" can either be opened for shared or exclusive access. A device is opened for shared or exclusive access on a first-come, first-serve basis. The guidelines are:
     - When a device is first opened for shared access, it can be opened for shared access by other programs.
     - A device that has been opened for shared access cannot be opened for exclusive access until all programs that have opened the device for shared access close the device.
     - When a device is first opened for exclusive access, it cannot be opened for either shared or exclusive access by another program. Once a device opened for exclusive access is closed, other programs can then open the device.
   - A program can receive input from both Xlib and Starbase, but can only use one method per each device/window combination.

The following table summarizes the input methods supported by X11.

Table 6-1. Input Methods Supported in an X11 Environment

XLIB/ Starbase Input Library	Usable in Raw Mode Starbase	Usable in an X11 Window	Exclusive/ Shared Access	Supported HIL Devices
Xlib	No	Yes	Shared	All HIL input
Starbase: SOX11 driver	No	Yes	Shared	X server pointer device and keyboard
Starbase: `libXwindow`	No	Yes	Shared	X server pointer
Starbase: `kbd`, `lkbd` driver	Yes	Yes	Shared or Exclusive	Terminal emulator `pty` files or ITE file
Starbase: HP-HIL driver	Yes	Yes	Shared or Exclusive	All HP-HIL devices except X server pointer and keyboard

Selecting the Right Input Driver or Library

The five input drivers and libraries suported in X11 windows are:

- Xlib library
- HP-HIL driver
- libXwindow library
- `kbd` and `lkbd` drivers
- SOX11 driver

The following table shows the advantages and disadvantages of these input drivers and libraries.

Table 6–2. Advantages and Disadvantages of the Input Driver and Libraries

Driver or Library	Advantages	Disadvantages
Xlib Library	■ Industry standard ■ NLS keyboard support ■ Supports toolkits(for example, HP widgets)	■ Must structure program to be event driven ■ Cannot receive input in virtual coordinates ■ Does not work in raw mode
HP-HIL Driver	■ Works in raw mode and X11 window	■ Cannot receive input from X11 pointer or keyboard ■ Does not provide NLS keyboard support
`libXwindow` Library	■ Can access X11 pointer ■ Supports toolkits (for example, HP widgets) ■ For local operation, code size is smaller than when using SOX11	■ Does not support raw mode input
`kbd` and `lkbd` Drivers	■ Works with any `tty/pty`	■ Not able to detect key releases ■ Does not support NLS
SOX11 driver	■ Supports Starbase input across the network ■ Provides input from X11 keyboard	■ Does not work in raw mode ■ When used for input, code size is large because of unused Xlib output code

Input Device/Window Combinations

The phrase "device/window combination" means that a program receives input from a certain device through a certain window. The only way to open a device for shared access in an X11 window is to request input from a device/window combination. Only one device/window combination can be active at one time. The following diagram and discussion will help clarify this concept. This diagram shows two programs attempting to get input from a single HIL tablet.

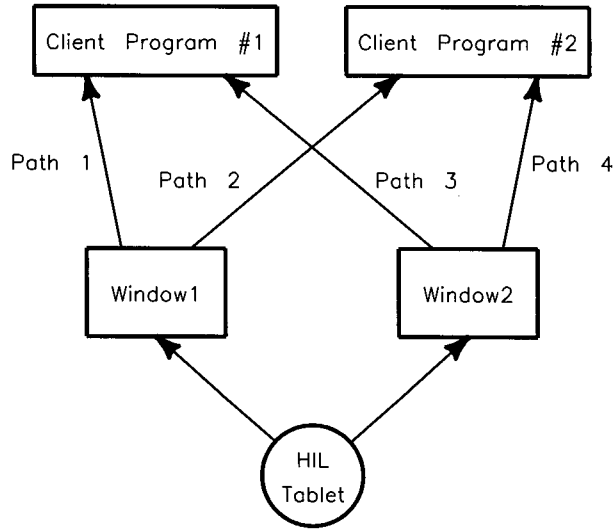


Figure 6-7. Input Device/Window Combinations

Main Points for the Previous Diagram

1. Path 1 and Path 2 represents a device/window combination consisting of the HIL tablet and Window 1. Likewise, Path 3 and Path 4 represents a different device/window combination consisting of the HIL tablet and Window 2.
2. Only one program can access a device/window combination at a time. In the above diagram, Paths 1 and 2 cannot be used at the same time. The same is true of Paths 3 and 4. However, Paths 1 and 4 (or 2 and 3) can be used when these Paths represent different device/window combinations.

The fact that Paths 1 and 2 (or 3 and 4) cannot be used at the same time is not usualy a hindrance since multiple programs will not attempt to obtain input from the same window.

3. Within a program, a given device/window combination can only be exercised by one driver at a time. For example, when Client Program 1 uses Xlib to obtain input through Window 1, it cannot also use the HP-HIL driver to obtain input from the same tablet through this same window. However, Client Program 1 could use the HP-HIL driver through Window 2 at the same time as Xlib because this represents a different device/window combination.

4. While a program cannot use both Xlib and Starbase input calls to obtain input from the same device/window combination, a program can use both Xlib and Starbase input calls in the same program and window to access different HIL devices. The different HIL devices ensure that the device/window combinations do not conflict.

Opening a Starbase Device/Window Combination

Starbase programs open device/window combinations using the gopen procedure. A particular device/window combination is specified by the ⟨*path*⟩, ⟨*kind*⟩, and ⟨*driver*⟩ parameters of the gopen procedure.

The following shows how the HP 300h device driver can be used to open a raw display, open a window created by the `xwcreate`(1) command, and open a window created by the `XCreateWindow`(3x) procedure.

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

`xcreate`:
```
fildes = gopen("/dev/screen/⟨windowname⟩",
OUTDEV, "hp300h", INIT);
```

`XCreateWindow`:
```
fildes = gopen("X11  hpsys:0.1  0x123456",
OUTDEV, "hp300h", INIT);
```

The above demonstrates three different syntaxes for the ⟨*path*⟩ parameter.

1. The syntax in the first example is that of a special device file and is denoted as a ⟨*special-device-file*⟩.
2. The syntax in the second example is that of a pty file created by the `xwcreate` command. This specifies a window and is denoted as ⟨*window-syntax*⟩.
3. The syntax in the third example is that of a path string which associates a display with the ID of a window created programmatically. This is also denoted as ⟨*window-syntax*⟩.

The following section describes how the ⟨*window-syntax*⟩ is used to open input devices. Input devices can be specified implicitly or explicitly.

Implicit Specification

Implicit specification means that the input device does not have to be specified explicitly but is implied through a combination of other parameters. The following are examples of implicit specification:

```
fildes = gopen("/dev/screen/windowname", OUTINDEV, "hp98550", INIT);
fildes = gopen("X11 hpsys:0.1 0x123456", OUTINDEV, "hp98550", INIT);
```

In both cases, the ⟨*path*⟩ parameter specifies ⟨*window-syntax*⟩ and denotes the window of interest. The ⟨*kind*⟩ parameter specifies that the window is being opened for both input and output. Whenever a window is opened for input using an output driver, the input is always obtained from the X11 server's pointer device. Therefore, the X11 server pointer device does not have to be referenced explicitly.

The fildes returned from the two gopen calls listed above works for output as well as input since the ⟨*kind*⟩ parameter is OUTINDEV. In this example, by opening the window as an OUTINDEV with a display driver, the input device is implicitly specified to be the X server's pointer device.

Another set of examples follows:

```
fildes = gopen("/dev/screen/windowname", OUTINDEV, "SOX11", INIT);
fildes = gopen("X11 hpsys:0.1 0x123456", OUTINDEV, "SOX11", INIT);
```

In the above, the Starbase-on-X11 (SOX11) driver is being used to open the window as both an input device and an output device. Starbase input calls obtain input from either the X11 server's pointer device or the X11 server's keyboard. Refer to "The Starbase-on-X11 Device Driver" in the *Starbase Device Drivers Library Manual* for details on input operation with the SOX11 driver. Note that neither the pointer device nor keyboard is referenced explicitly.

The table shows the devices that are implictly specified when a ⟨*window-syntax*⟩ ⟨*path*⟩ parameter is used.

Explicit Specification

In the case of explicit specification of an input device, the ⟨*path*⟩ parameter identifies both a window and an input device. The following examples show how the first tablet on the HIL loop is opened by the Starbase HP-HIL driver for input through a window.

```
fildes = gopen("/dev/screen/windowname  FIRST_TABLET", INDEV, "hp-hil", INIT);
fildes = gopen("X11 hpsys:0.1 0x123456  FIRST_TABLET", INDEV, "hp-hil", INIT);
```

In these examples, the ⟨*path*⟩ parameter of the gopen procedure defines the combination of a particular window and a particular device (FIRST_TABLET). The string FIRST_KEYBOARD is a specific example of a general syntax used to specify an HIL device. This is denoted as the ⟨*device-syntax*⟩ and is defined as follows:

⟨*device-syntax*⟩ = ⟨*position*⟩_⟨*device-type*⟩

where ⟨*position*⟩ represents the position of the device on the interface loop (FIRST, SECOND, THIRD, etc.) and is determined by following the HIL cable from the workstation to the device and counting how many devices there are of the same type. ⟨*device-type*⟩ is a name that specifies the type of device.

The ⟨*device-type*⟩ must be one of the following names:

- MOUSE
- TABLET
- KEYBOARD
- BUTTONBOX
- ONE_KNOB
- NINE_KNOB
- TRACKBALL
- QUADRATURE

X11 supports more device types than shown above but these are the only valid device types for use with Starbase.

For example, when a workstation is configured with a keyboard and two graphics tablets, connected in that order, the valid ⟨*device-syntax*⟩ strings are:

- FIRST_KEYBOARD
- FIRST_TABLET
- SECOND_TABLET

The following table specifies the values of the gopen parameters to use when opening various devices using the Starbase input drivers that are supported in X11 windows.

Table 6-3. gopen **Parameters for Starbase Input in an X11 Window**

Input Library	Devices Accessed	Shared/ Exclusive	gopen Path Parameter	⟨*kind*⟩ Parameter	⟨*driver*⟩ Parameter
SOX11	X server pointer	shared	⟨*window-syntax*⟩	INDEV or OUTINDEV	SOX11
	X server keyboard	shared	⟨*window-syntax*⟩	INDEV or OUTINDEV	SOX11
lib-window	X server pointer	shared	⟨*window-syntax*⟩	INDEV or OUTINDEV	hp98550, hp300h, etc.
kbd, lkbd	X server keyboard	shared	⟨*pty-pathname*⟩	INDEV	kbd, lkbd
	Other Keyboards	exclusive	dev/console	INDEV	kbd, lkbd
HP-HIL	Other Keyboards	shared	⟨*device-syntax*⟩	INDEV	hp-hil
	Other HIL Devices	shared	⟨*device-syntax*⟩	INDEV	hp-hil
	Other Keyboards	exclusive	⟨*special-device-file*⟩	INDEV	hp-hil
	Other HIL Devices	exclusive	⟨*special-device-file*⟩	INDEV	hp-hil

Starbase Input from the X11 Pointer Device

The SOX11 driver and the `libXwindow` library permit Starbase programs to receive input from the pointer device through the X11 server. This input consists of both locator and choice input and is identical for both the SOX11 driver and the libXwindow library.

The `libXwindow` library is used by display drivers (for example, the HP 98550 device driver) to receive input from the X server's pointer device. It is analogous to the `libwindow` library, used by programs in an HP Windows/9000 environment. The `libXwindow` library provides the procedures that allows Starbase display drivers to access the X11 pointer device.

The HP Two-Button Mouse

Because of the mouse-oriented nature of the X Window System, the HP X11 server treats the two-button mouse as having three buttons; the third ("middle") button goes down when both buttons are pressed at the same time. To be consistent with X11, the SOX11 driver and `libXwindow` library treat the two-button mouse as having three buttons.

Details of Starbase Input from the X Server's Pointer Device

Starbase supports the following four types of input; refer to *Starbase Graphics Techniques* for a description of each of these types:

- Sampling
- Requests
- Tracking
- Events

Furthermore, Starbase supports two classes of input devices:

- Locator Devices
- Choice Devices

Locator and Choice Ordinals

When opened by the `libXwindow` library, the X pointer provides one Starbase locator ordinal and two Starbase choice ordinals. The SOX11 driver also provides one locator ordinal, but provides three Starbase choice ordinals; the additional choice ordinal is used for keyboard input.

The locator ordinal provides pointer position information. Location information is returned in Starbase virtual device coordinates (VDC's).

The two choice ordinals provide information about the pointer's buttons. The choice ordinals are different formats of the same information. The definitions of the two choice ordinals are listed below (note the similarity with the HP-HIL driver and HP Windows/9000 libwindow library):

- Choice ordinal 1: This selection returns button numbers 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. If the `TRIGGER_ON_RELEASE gescape` is not enabled, or if the `IGNORE_RELEASE gescape` is enabled, a button release causes choice ordinal 1 to return zero.
- Choice ordinal 2: This selection returns a 32-bit-wide bitmap. The least significant bit equals button 1 and the most significant bit equals button 32. A value of 1 in a bit indicates that the corresponding button is currently pressed. Conversely, a value of zero in a bit indicates that the corresponding button is currently released.

Receiving Input When the Pointer Device is Outside of the Window

When the pointer device moves outside of the window, the pointer location returned by Starbase is the position on the window boundary where the pointer device exited the window.

A program can continue to receive input from the pointer even when the pointer is out of the window when:

1. Initially, all buttons are up.
2. One or more buttons are pressed while in the window.

3. One or more of the buttons is still down, and the pointer is moved out of the window.
4. The program continues to receive pointer input until all buttons are released.

In X terminology, this is called a grab. Another effect of grabs is that sometimes a program will not receive button press/release information. Some X programs use grabs to keep other programs from receiving pointer information. For example, when a window manager brings up a menu, it typically grabs the pointer until the user is done selecting a menu item.

Starbase Sampling of the X11 Pointer

Locator and choice sampling of the X11 pointer device operates as follows:

1. The X server's locator position can be sampled anytime, and is returned relative to the window. By default, when the X pointer is on a different screen than the window, the ⟨*valid*⟩ parameter of the `sample_locator` procedure is returned as `FALSE`. When the `gescape OLD_SAMPLE_ON_DIFF_SCREEN` is used, the ⟨*valid*⟩ parameter is returned as `TRUE` when the X pointer is on a different screen. In this case, the pointer position returned by `sample_locator` is either the last value of the X locator on that screen or the value (0,0) if the pointer has never been on that screen. To restore the default (⟨*valid*⟩ set to `FALSE`) behavior use the `BAD_SAMPLE_ON_DIFF_SCREEN gescape`.
2. The pointer's button(s) can be sampled anytime and always return valid values.
3. Choice ordinal 1 contains the number of the last button pressed when a button is still down, or zero when no buttons are down. If the `TRIGGER_ON_RELEASE` gescape is enabled, button releases cause the value to be a negative button number.

Starbase Tracking of the X11 Pointer

Starbase tracking of the X11 pointer device operates as follows:

1. The X pointer may be tracked to the same window or to any other window or display.
2. When tracking is enabled, the X cursor will not be visible while over the output window (so as not to interfere with the Starbase echo). The Starbase tracking process will move the Starbase echo according to the movement of the X pointer while the pointer is over the window.
3. When tracking is turned off, the X cursor will once again be visible while over the window and the Starbase tracking daemon will no longer move the Starbase echo.

Starbase Requests and Events with the X11 Pointer

Starbase requests are satisfied and caused by "triggers." The details of Starbase triggers with the X11 pointer are as follows:

1. Triggers are button presses or releases while the X locator is over the unobscured portions of the window. Also, when a button press causes the Starbase application to grab the pointer, until all buttons are released, other button presses or releases will also be triggers.
2. Triggers do not include button presses or releases which are grabbed by another X client (e.g., a window manager).

Starbase Input from Non-Pointer Devices

The previous section covered Starbase input from the X11 pointer device; this section covers Starbase input from the three remaining groups of devices:

- X Server Keyboard
- Other Keyboards
- Other HIL Devices

These three groups of devices are referred to as the "non-pointer devices." The HP-HIL and SOX11 drivers provide input from the non-pointer devices. The Starbase `kbd` and `lkbd` drivers are not discussed because they receive input from a terminal emulator `pty` file in contrast to the above drivers which receive input directly through a graphics window. The discussion of HP-HIL is for shared access, not exclusive access.

Keyboard Input

The keyboard input values returned by the Starbase HP-HIL driver are different than the keyboard input provided by the SOX11 driver. The SOX11 driver provides NLS support and uses X11 key definitions. The HP-HIL driver does not provide NLS support and ignores the user's X11 key definitions. The HP-HIL driver returns the same values for raw mode Starbase and Starbase in an X11 window. Refer to the *Starbase Device Drivers Library Manual* for more information on the type of input returned by each driver.

Starbase Sampling, Requests, Tracking and Events

The following table shows which input drivers can be used to access locator and choice devices for each of the four types of Starbase input.

Table 6-4. Input Drivers Used to Access Starbase Locator and Choice Devices

Type of Input Device	Sampling	Tracking	Requests	Events
Locator Device (Other HIL Devices)	HP-HIL Driver	HP-HIL Driver	HP-HIL Driver	HP-HIL Driver
Choice Device (X11 Keyboard & Other Keyboards)	HP-HIL Driver SOX11 Driver X11kbd Driver		HP-HIL Driver SOX11 Driver X11kbd Driver	HP-HIL Driver SOX11 Driver X11kbd Driver

Details of Starbase Sampling of Non-Pointer Devices

Starbase locator sampling with the non-Pointer devices works as follows:

- Locator and choice devices can be sampled anytime. New data is not normally provided if the input device's window is out of focus. The exception is when a button press causes a grab to occur. Then, the program continues to receive input after the window goes out of focus, until all buttons have been released on the device.
- If the window is out of focus, the ⟨*valid*⟩ parameter of `sample_locator` is returned as `TRUE` and the locator position returned by `sample_locator` is either the last value of the locator when that window was in focus, or the value $(0, 0)$ if the window has never been in focus.
- When a window goes out of focus and later comes back into focus, the locator or choice value is the same as it was when the window went out of focus.
- Choice ordinal 1 will contain the number of the last button pressed when a button is still down, or zero when no buttons are down. If the

TRIGGER_ON_RELEASE gescape is enabled, button releases cause the value to be a negative button number.

Tracking Non-Pointer Devices

The details of tracking Starbase locator devices are as follows:

- The locator device may be tracked to the same window or any other window or display.
- When tracking is turned on and the window is in focus, the Starbase tracking process will move the Starbase echo according to the movement of the locator.
- When a window goes out of focus and later comes back into focus, the locator value will be the same as it was when the window went out of focus.
- When tracking is turned off the Starbase tracking process will no longer move the Starbase echo.

Starbase Input Examples

This section provides examples to demonstrate some typical changes that are required to make a raw mode program work in an X11 window.

Example 1: Application that uses a Tablet and a Button Box

Assume that you have a program called "example" that runs in a raw environment and receives input from a tablet and a button box. You want to be able to run your program in an X11 window. Also assume that the ⟨*path*⟩ and ⟨*driver*⟩ parameters of the gopen procedure calls are passed into the program via the following environment variables:

- TABLET_DEVICE
- TABLET_DRIVER
- BUTTONBOX_DEVICE
- BUTTONBOX_DRIVER

No source code modifications are required to run the program in an X11 window Because the parameters of the gopen statement are passed into the program. The program only needs to be linked with the correct libraries, have its environment variables set to the correct values, and run. The following is an example link sequence where the program will only be run on an HP 300h device:

```
cc -o example example.o -ldd300h -lddhil -lXwindow -lsb1 -lsb2          \
  -lXhp11 -lX11 -lm
```

Use the xwcreate procedure to create the window. The following example creates a window called "example_window":

```
xwcreate example
```

Assume that the 32-button box is not used as the X server's keyboard device. The correct values of the buttonbox environment variables are:

```
BUTTONBOX_DEVICE = "/dev/screen/example_window FIRST_BUTTONBOX"
BUTTONBOX_DRIVER = hp-hil
```

When the tablet is the server's pointer and the window that the program runs in is called "example_window", set the tablet environment variables to the following values:

```
TABLET_DEVICE = "/dev/screen/example_window"
TABLET_DRIVER = hp300h
```

Otherwise, the tablet is an extra (OTHER-HIL) device, and the environment variables should be set to:

```
TABLET_DEVICE="/dev/screen/example_window FIRST_TABLET"
TABLET_DRIVER=hp-hil
```

Finally, start the program!

Example 2: Application That Works in an HP Windows/9000 Environment

A Starbase program can receive input from the window system pointer device from both HP Windows/9000 and X11. A window-dumb application that already works in a Windows/9000 environment and only receives input from the Windows/9000 locator device will work in an X11 environment with no source code changes (because of the input code). Even if the ⟨*path*⟩ parameter of the gopen procedure is hard-coded, you only need to create a window of that name using the `xwcreate` command.

Example 3: Application Using the SOX11 Driver

The SOX11 driver allows an application to run in a window on a remote host, while the Starbase display drivers (for example, the HP 98550 device driver) support Starbase in a local X11 window. It is straightforward to write an application that either works in a local window using a display driver or in a remote window using the SOX11 driver. Note that a problem may occur if the program uses Starbase features not supported by the SOX11 driver. Refer to the chapter "SOX11 Device Driver" of the *Starbase Device Drivers Library Manual* for more information.

Assume that the program requires input from the X server's pointer device and a button box. The following guidelines should be followed:

- To open both the window and the device, the gopen procedure is used to open the window as an `OUTINDEV`.
- To open the button box, the gopen procedure is used to open the button box with the HP-HIL driver using the ⟨*device-syntax*⟩ ⟨*path*⟩ parameter.

Example Code Segment

```
main(argc, argv)
int     argc;
char    *argv[];
{
    int         window_fildes, buttonbox_fildes;

    if (argc < 4) {
        printf("Sorry, not enough arguments!\n" );
        printf("Usage:\n %s %s %s \n",
               "pgm_name",
               "window out_driver",
               "button_box" );
        return(1);
    }
    window_fildes = gopen(argv[1], OUTINDEV, argv[2], INIT);
    buttonbox_fildes = gopen(argv[3], OUTINDEV, "hp-hil", INIT);
      :
    gclose(window_fildes);
    gclose(buttonbox_fildes);
}
```

Additional Guidelines for Device/Window Combinations

This section assumes you understand Xlib input, particularly passive grabs. for information on how passive grabs work, refer to the chapter "Window Manager Functions," in *Programming With Xlib, Version 11.*

There are cases where it is advantageous for an application to use both Xlib and Starbase calls to receive input from the same device/window combination.

For example, suppose that you desire to enhance a Starbase application to use Xt Intrinsics and HP Widgets to provide three-dimensional menus. One alternative is to have a large (main) window with a sub-window for Starbase to run inside. The toolkit code can work inside other sub-windows of the main window. In this way, the toolkit code and Starbase each use different windows to talk with the X server's pointer. The only disadvantage to this is that the user can never bring up a widget (that is, the menu) by pressing a button in the Starbase window.

Another approach is to bring up a menu from a button press in a Starbase window, using passive grabs to make sure that programs see button releases for which a button press was seen. Only one program or Starbase driver can establish a passive grab per device/window combination. Passive grabs are granted by the X server according the window hierarchy tree. The program could use Xlib to establish a passive grab on the program's main window, which is an ancestor of the Starbase window. Then, when the user presses a button in the Starbase sub-window, the X server first offers the button press to the Xlib connection. If the program's response to that press is to pop up a menu, the toolkit code could be called. If not, the program can replay the event and then Starbase would see the button press. In this way, the rule of one open or connection per device/window combination is followed, but the program appears to have two open connections to the same device/window combinations.

For more information, refer to the manual *Programming With Xlib, Version 11.*

7

Graphics Hardcopy Operation

The term **graphics hardcopy** refers to the ability to print an image from a display or a file to a graphics printer, as shown in the following diagram:

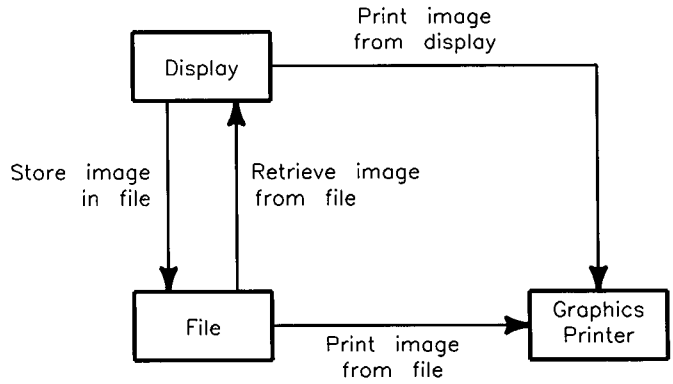


Figure 7–1. Graphics Hardcopy from a Display or File

The image can be printed either directly from the display or from the file.

Graphics Printers versus Vector Plotters

Graphics hardcopy is performed by copying the pixels on the display to dots on the printer. Such printers are often called graphics printers or raster printers and are distinct from plotters which use pens to draw lines; these are typically called vector plotters or pen plotters.

While the image on your display may be composed of many vectors, the graphics hardcopy procedures described in this chapter do not permit these vectors to be copied from your display to a vector plotter. However, if you are using Starbase you can open a Starbase driver (for example, the Starbase `HPGL` driver), which interfaces to vector plotters. You can then direct the output to a vector plotter instead of the display.

Starbase and X11 Hardcopy Documentation

Both Starbase and X11 provide graphics hardcopy capabilities, procedures (library subroutines), and utilities (user commands). Starbase's capabilities are described in the chapter "Storing, Retrieving and Printing Images" in *Starbase Graphics Techniques.* X11 hardcopy capabilities are described in *Programming With Xlib, Version 11.* Note that neither X10 nor X11 revision A.00 support graphics hardcopy capabilities.

Procedures and Commands

The following diagram shows the relationship of X11 and Starbase hardcopy procedures and utilities.

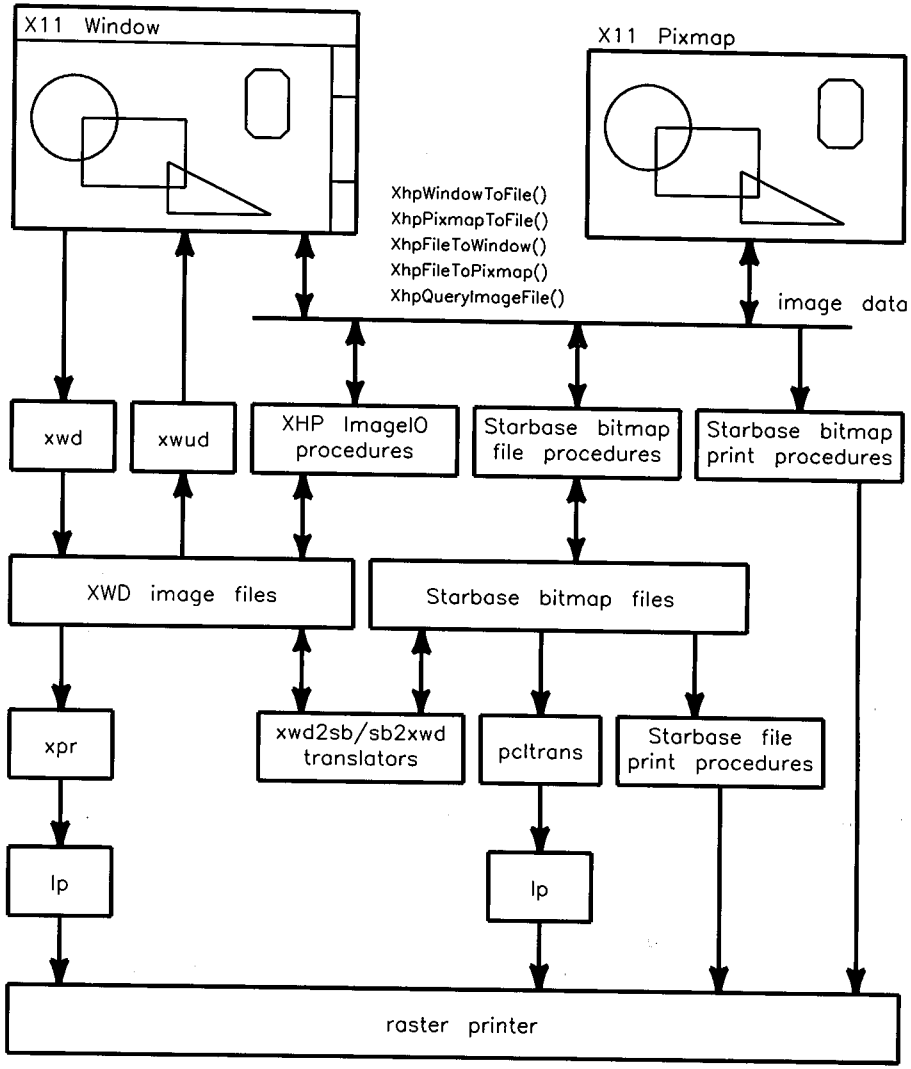


Figure 7–2. Relationship of X11 and Starbase Hardcopy Procedures and Utilities

Notes on the Previous Diagram

- The supported file formats for the raster data are the X11 and Starbase formats. "XWD format" denotes X11 files, where XWD stands for "X11 Window Dump." Starbase files are denoted as Starbase bitmap files. Procedures are provided to translate from XWD format to Starbase format (`xwd2sb`) and from Starbase format to XWD format (`sb2xwd`).
- Printing with X11 always requires the display image to be stored in the XWD file first. Starbase procedures are provided to do a graphics hardcopy directly from the display as well as from a file.
- Four ways to print an image are:
    1. Use the X11 command `xpr` to print XWD format files, which can then be piped into the `lp` spooler for output to the printer.
    2. Use the `pcltrans` command to print Starbase bitmap files, which pipe into the `lp` spooler for output to the printer.
    3. Use Starbase procedures within a program to output Starbase bitmap files to the printer.
    4. Use Starbase procedures within a program to print hardcopy directly from the display to a printer.
- The `xwd` and `xwud` commands store and retrieve the entire window. The `XHPImageIO` procedures permit storage of a subrectangle of the window and provide access to pixmaps. For example, the `XHPWindowToFile` procedure supports an x and y start position, and length and width parameters to specify the subrectangle to be stored in the XWD file.
- The `xwd` and `xwud` commands were provided with the X11 revision A.00 server; the `XHPImageIO`procedure, and the `xpr`, `xwd2sb`, and `sb2xwd` commands were first introduced with the X11 server.
- The `xwd` and `xwud` commands work only within an X11 window, not from an X11 pixmap. The `XHPImageIO` procedure works on both windows and pixmaps.

Method Selection

Several factors must be considered when determining whether to use the X11 or Starbase method:

- In general, the X11 command `xpr` is simple and quick while the Starbase hardcopy methods are more sophisticated but slower. The `xpr` command is most appropriate for hardcopy of text and simple graphics. For example, `xpr` is best for doing a hardcopy of a terminal emulator window, while Starbase hardcopy is best for complex graphics such as shaded 3D images. The dithering/diffusion process of Starbase hardcopy introduces artifacts which reduce text readability. Thus, Starbase hardcopy is not recommended for applications such as generating hardcopy of a terminal emulator window.
- When your application requires image file interchange, you should consider using the X11 dump/hardcopy procedures since the XWD file format is more widely used than is the Starbase file format.
- Because the X11 hardcopy procedures and utilities are executed by the client, the graphics printer must be connected to the client. When the client happens to be a remote computer (that is, connected to the server over the network), the pixels to be printed have to first be moved from the server to the client over the network. This is time consuming and can degrade network performance. Therefore, where possible, your client application doing the graphics hardcopy should be run locally.
- When using Starbase remotely over the network via the SOX11 driver, you can use the Starbase hardcopy procedures and utilities on a remote computer. In this case, the graphics printer would be connected to the same computer running the SOX11 driver. This has the disadvantage that pixels are transferred over the network. A solution is to open the same X11 window that the SOX11 driver is drawing to and then use local-only Starbase hardcopy procedures. The remote client must communicate the window's ID to the local client to open the correct window.
- When you want to do a graphics hardcopy of the entire display, you can specify the root window.

8

Program Development Guidelines

Starbase in X11 Windows

Reading Chapter 3 prior to reading this chapter will help you understand the program.

Several topics are discussed in this chapter:

- The degree of Starbase source and object code compatibility provided with the 3.1 and 6.5 releases of HP-UX.
- The libraries that need to be linked with your program.
- The guidelines to be followed in developing your application.
- Tips on porting a Starbase application from the following environments to X11.
    - HP Windows/9000
    - X10
    - X11

Source and Object Code Compatibility

In order for a Starbase program to work in an X11 window, it must be linked with the Starbase libraries provided with the 3.1 (Series 800) and 6.5 (Series 300) releases of HP-UX (or later releases). The X11 server must be 3.1/6.5 (revision A.01) or later and must be from the same or later release as Starbase.

Source Code Changes

The only source code change potentially required for a window-dumb program to operate in an X11 window is to modify the Starbase **gopen** statement to permit the parameters to be passed in at run time.

Unlinked Object Code Compatibility

A pre-3.1/6.5 compiled Starbase program can be linked with the 3.1/6.5 Starbase libraries and executed in an X11 window, assuming the **gopen** parameters are passed into the program at run-time. You may get a floating-point warning from the linker, but this can be ignored.

Pre-3.1/6.5 Linked Object Code Compatibility

Pre-3.1/6.5 Starbase programs will work with the 3.1/6.5 (or later) releases of HP-UX in raw mode but will not run in an X11 window.

Linking Your Program

The 3.1/6.5 software release Starbase drivers operate in these environments:

- Raw mode
- HP Windows/9000
- X11

Starbase Link Sequence

The order in which the Starbase libraries are linked is shown below. Libraries which are optional (depending on your application) are marked with a "*".

1. Your *program*.c
2. Appropriate Starbase input drivers and/or output drivers (in any order).
3. * Starbase bit and/or byte drivers, `-lddbyte`, `-lddbit` (in either order). These are required for Starbase backing store.
4. * `-lwindow`. This is required to run Starbase in a Windows/9000 window.
5. * `-lXwindow`. This is required to run Starbase in an X11 window.
6. * `-lsbdl`. This is required if the program uses the Starbase display list.
7. `-lsb1`
8. `-lsb2`
9. * `-lXr`. This is required to use the Xray library in your program.
10. * `-lXw`. This is required to use X Widgets in your program.
11. * `-lXt`. This is required to use the X Toolkit in your program.
12. * `-lXhp11`. This is required if your program is intended to execute in an X11 window, regardless of whether your program is window-dumb or smart.
13. * `-lX11`. This is required if your program is intended to execute in an X11 window, regardless of whether your program is window-dumb or smart.

Example link sequences are provided in the following sections. The Starbase driver `/usr/lib/libdd98550.a` is used in these examples.

Raw Mode Starbase Operation

The link sequence for raw operation is:

```
cc -o prog prog.o -ldd98550 -lsb1 -lsb2
```

Starbase Operation in HP Windows/9000

The link sequence for a Starbase program to operate in HP Windows/9000 is:

```
cc -o prog prog.o -ldd98550 -lwindow -lsb1 -lsb2
```

As with pre-3.1/6.5 HP Windows/9000 operation, the library `libwindow` must be linked in. A Starbase program linked as above can still be run in raw mode.

Xlib Operation

The link sequence for a non-Starbase (`Xlib` only) program is:

```
cc -o prog prog.o -lXhp11 -lX11
```

Refer to *Programming With Xlib, Version 11* for more details.

Starbase Operation

The link sequence for a Starbase program to operate in X11 windows is:

```
cc -o prog prog.o -ldd98550 -lXwindow -lsb1 -lsb2 -lXhp11 -lX11
```

The `libXwindow` library permits Starbase to perform necessary communications with the X11 server (for example, obtaining X clip lists, or permitting Starbase to get input from the X pointer device). Note that a program linked as above can still be run in raw mode.

Using Both Libraries

`libXwindow` and `libwindow` can both be linked into your application. This will permit the application to run in raw mode, in HP Windows/9000 and in X11 windows. For example:

```
cc -o prog prog.o -ldd98550 -lwindow -lXwindow -lsb1 -lsb2 -lXhp11 -lX11
```

Starbase Retained Rasters (Backing Store)

To support Starbase retained raster (backing store), the bit and/or byte driver must be linked as shown below. Because backing store only applies to window systems, linking the bit and/or byte driver is only necessary when your application is intended for operation within a window and you intend to use backing store. Again, both the `libXwindow` and `libwindow` drivers can be included, as shown below:

```
cc -o prog prog.o -ldd98550 -lddbyte -lddbit -lwindow -lXwindow          \
  -lsb1 -lsb2 -lXhp11 -lX11
```

The bit driver `/usr/lib/libddbit.a` (`-lddbit`) is used to provide backing store for monochrome displays. The byte driver `/usr/lib/libddbyte.a` (`-lddbyte`) is used to provide backing store for color displays.

The bit and byte drivers are only used to provide backing store for Starbase. The X11 server already includes the necessary routines to support backing store for Xlib applications without linking in the bit and byte drivers.

Starbase Drivers to Link Into Your Application

Many application developers find it convenient to develop one executable with all supported drivers linked into the application. The end user then typically passes in the Starbase gopen parameters to cause the program to access the desired display. The alternative to one large executable with all drivers linked in is separate executables, one for each display. This leads to smaller individual executables but a greater number of them.

Application Development Guidelines

This section provides application development guidelines to assist in developing Starbase programs which run in an X11 window.

- To ensure maximum Starbase performance, Starbase should be operated in buffered mode. This is not related to double buffering, but refers to the ability of Starbase to buffer multiple operations before obtaining a lock on the display.
- Starbase `make_picture_current` calls should be kept to a minimum since they flush the buffer and thus reduce performance by reducing the effective buffer size.
- It is possible to have name conflicts in header files. For example, the Starbase header file contains this line:

  ```
  #define  line_width  c_line_width
  ```

  Likewise, the header file `X11/Xlib.h` defines `line_width` to be a member of the `XGCValues` structure. If your program contains both header files and refers to the `line_width` member of the `XGCValues` structure, you will get a fatal compile-time error. You may be able to work around this by using a seperately compiled procedure of your own to hide a conflicting procedure. For example, you could create a `my_line_width` procedure which calls `line_width` and compile this separately.
- It is common for a raw mode Starbase program to do a Starbase open of an HIL mouse to get locator input. This same program will not work in an X11 window because the X11 server grabs the mouse for its pointer device. Therefore, the Starbase program should get its locator input from the pointer device mouse using the `libXwindow` library by changing the gopen parameters.
- Use the utility `/usr/lib/X11/xwininfo` to get the window's ID for use in a Starbase open of an existing window (for example, a terminal emulator window). When executed with the `-tree` option, it permits you to obtain the window ID of all existing windows on your display. The window ID can then be used in formulating your Starbase `gopen` ⟨*path*⟩ string.
- Use backing store sparingly, as it degrades performance.

Moving HP Windows/9000 to X11

For a information on converting your window system from HP Windows/9000 to X11, refer to *HP Windows/9000 to X Window System Conversion Guide.*

Window Types

HP Windows/9000 defines both a graphics window type and a terminal window type. Only the graphics window type can be used for Starbase output. X11, however, defines one type of window which can be used for both graphics and terminal output.

Input

The Starbase `libXwindow` library permits input to be received from the X11 server's pointer. The pointer device is always opened for shared access. This library provides Starbase input similar to the HP Windows/9000 `libwindow` library.

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 software release Starbase drivers so that the program operates directly in an X11 window. This will greatly increase performance, but remote operation will be lost.

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 to 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 software release Starbase drivers so that the program operates directly in an X11 window. This will greatly increase performance, but remote operation will be lost.

Glossary

Backing Store
Memory used to store graphics data that is rendered in obscured portions of the window.

Bit Driver
Used to provide backing store for monochrome displays.

Byte Driver
Used to provide backing store for color displays.

Client
A program which needs the services and processes of a server machine in order to run.

Color Map Policy
A mechanism used to control the interaction of Starbase and Xlib color map procedures.

Combined Mode
The X11 server operates in both the overlay planes and image planes simultaneously.

Cursor
The graphics entity which follows the X11 pointer device. Also called *X11 cursor.*

Display Control Data
Data that affects what is seen on the entire display, but does not affect the actual data stored in the frame buffer.

Display Control Focus Window
The window where rendering appears valid. Because only one software color map can be downloaded into hardware at a time, the other windows

use the display control focus window's color map, which is in hardware, to render their displays.

Display Control Policy
Specifies how the display control data is handled.

Drawing Control Data
Data that controls what is actually written into the frame buffer. An example is the replacement rule.

Echo
The graphics entity controlled by Starbase; can be "attached" to an input device. Also called *Starbase echo.*

Focus Policy
The policy which controls how the input focus changes from one window to another.

Font Character Set
Specifies the raster patterns of characters displayed on the screen.

Font Files
Files on disc which contain actual raster text patterns.

Font File Format
The disc format the font is stored in.

Graphics Hardcopy
The ability to print an image from a display or file to a graphics printer.

Hardware Color Map
Provides the actual physical translation between a pixel value in the frame buffer and the color generated on the screen. The display control focus window's software color map is downloaded into the hardware color map, so there is only one hardware color map but can be numerous software color maps.

Image Mode
The X11 server operates only in the image planes.

Input Focus
The ability to direct input to one window at a time in X11.

Input Focus Window
The window that receives input events.

Local Operation
Operating a program on the same computer as the X server.

Network Transparency
Can run a program as either a local operation or a remote operation without re-writing the program.

Overlay Mode
The X11 server operates only in the overlay planes.

Raster Text
Text which is represented by patterns of individual pixels which form a character on the display. Compare *stroke text.*

Raw Mode
Indicates that a window system is not running on the display. In raw mode, a graphics application typically has control of the entire display.

Raw Mode Operation
A program which is operating in raw mode, i.e., no window system is being run with the program.

Remote operation
Operating a program across a network on another computer. For example, a remote client operates on a different computer than the X server.

Retained Window
A window that has backing store associated with it.

Root Window
The window where the X11 system is started up.

Server
A computer which provides services and processes to a client program.

Software Color Map
The program's representation of the color map; there can be numerous software color maps.

Stacked Screen Mode
The X11 server operates in the overlay and image planes as two separate screens.

Starbase Echo
The graphics entity controlled by Starbase; can be "attached" to an input device. Also called *echo*.

Stroke Text
Text which is represented by a combination of short vectors. Compare *raster text*.

Window-Smart Program
A program that uses window library calls. A window smart program can only be run within a window.

Window-Dumb Program
A program which does not make any window calls. Such a program can typically be run both in raw mode and within a window.

Window System
A package that includes the user interface, any available clients, the library of routines that manipulate the windows, and the server.

Window System Pointer Device
The device used to point to the currently desired window for an operation; a mouse is typically used.

X11 Cursor
The graphics entity which follows the X11 pointer device. Also called *cursor*.

X Window System
A window system whose characteristics are common to X10 and both revisions of X11 (revision A.00 and A.01).

A

Documentation Bibliography

Introduction

The following are references you may find useful when working with the given topics:

AGP/DGL Documentation

- *AGP Programmer's Manual*
- *DGL Programmer's Manual*
- *DGL/AGP Device Driver's Manual*

Fast Alpha/Font Manager Documentation

- *Fast Alpha Font Manager Programmer's Manual*

HP-GKS Documentation

- *HP-GKS Device Drivers Library*
- *HP-GKS FORTRAN Pocket Reference*
- *HP-GKS User's Guide*

HP-UX Documentation

- *HP-UX Software Catalog* published by Technical Systems Sector
- *HP-UX System Administrator Manual*

HP Windows/9000 Documentation

- *HP Windows/9000 Programmer's Manual*
- *HP Windows/9000 Reference*
- *HP Windows/9000 to X Window System Conversion Guide*

Starbase Display List Documentation

- *Starbase Display List Programmer's Manual*

Starbase Documentation

- *A Beginner's Guide to Using Starbase*
- *Starbase C Pocket Reference*
- *Starbase Device Driver's Library Manual*
- *Starbase FORTRAN Pocket Reference*
- *Starbase Graphics Techniques*
- *Starbase Pascal Pocket Reference*
- *Starbase Reference Manual*

X10 Documentation

- *Getting Started With the X Window System*
- *Programming with the X Window System*
- *X10 Xlib Programming Manual*

X11 Documentation

- *Beginning User's Manual*
- *Configuring the X Window System*
- *Programming with the HP XWidgets and XIntrinsics*
- *Programming with Xlib, Version 11*
- *Programming with Xrlib*
- *Using the X Window System, Version 11*
- *X11 Programming Manual* by O'Reilly
- *X11 Reference Manual* by O'Reilly
- *Xlib Quick Reference Guide*

Index

E

F

G

H

I

P

R

S

T

V

W

X

Z

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 Programming with X11

	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 1/1/1991. (Manual: 98592-90000 E1288)

Please Tape Here

Please print or type your name and address.

Name: _____

Company: _____

Address: _____

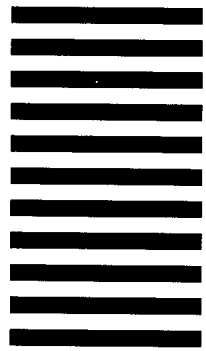
City, State, Zip: _____

Telephone: _____

Additional Comments: _____

Starbase Programming with X11
HP Part Number 98592-90000
E1288

NO POSTAGE
NECESSARY
IF MAILED
IN THE
UNITED STATES



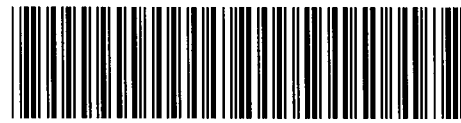
BUSINESS REPLY MAIL

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
98592-90000
Microfiche No. 98592-99000
Printed in U.S.A. E1288



98592-90636
For Internal Use Only