HEWLETT PACKARD

Manual Update

This Manual Update is for Reorder Number: 98592-90000
Title: Starbase Programming with X11
Printing Date: 12/88

The purpose of this manual update is to provide new information for your manual to bring it up to date. This is important because it ensures that your manual accurately documents the current version of the product.

This update consists of this cover sheet, a printing history page and all replacement pages. Replacement pages are identified by a revision date at the bottom of the page. A vertical line (change bar) in the margin indicates new or changed text material. The change bar is not used for typographical or editorial changes that do not affect the technical accuracy. New pages to be added do not contain change bars.

To update your manual follow the instructions below, replace existing pages with the Update pages and insert new pages as indicated. Destroy all replaced pages.

For all pages in this update, discard the corresponding pages in the manual, replacing them with the new pages.

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

HP Part Number
98592-90818
Printed in U.S.A. U0289

98592-90818

Starbase Programming with X11

HP 9000 Series 300/800 Computers

HP Part Number 98592-90000

HEWLETT
PACKARD

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

Notices

Printing History

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

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

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.

February 1989 ... Update to incorporate new information and correct technical errors.

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

Contents

Window-Smart Program Development

The previous sections described how to take a window-dumb program and direct it to run in an X11 window. An alternative to a window-dumb program is a "window-smart" program. A window-smart program makes Xlib window calls. It can create windows, open windows for graphics output, destroy windows, use the Xlib library for output and input, etc.

This section provides guidelines to help you select between window-smart and window-dumb program development. Subsequently, guidelines are given for window-smart development. Again, this section does not cover the more complex interactions between X11 and Starbase. These interactions are discussed in subsequent chapters.

Selecting Between Window-Smart or Window-Dumb Programs

A fundamental decision you'll need to make is whether your application program should be X11 window-smart or X11 window-dumb. Both approaches have benefits and limitations. As discussed previously, the primary advantage of a window-dumb program is that it can be run in raw mode or in an X11 window. However, a window-dumb program cannot take advantage of the powerful capabilities provided by X11 windows.

The primary advantage of developing a window-smart graphics application program is that you have programmatic control of not only the graphics but the window system as well. You can create windows, move windows, use Xlib input and output primitives, etc. You can also mix Starbase calls and Xlib calls in the same program to make optimum use of the available hardware and software resources. A window-smart program, however, cannot be run in raw mode. Because Xlib calls are not supported in raw mode, a window-smart program can only run within a window.

It is expected that many application developers will chose to develop window-smart programs for the following reasons:

- Window-smart applications have access to the powerful capabilities provided by X11. Because X11 is an industry standard, using Xlib will increase program portability to different platforms.
- As workstation graphics and CPU performance increases, multiple applications, each executing within its own window, will become the norm. Application developers will focus less on raw mode operation (where they consume the entire display) and more on operation within a window.

Sequence for a Window-Smart Program

This section gives a brief overview of the sequence a window-smart program might follow. To create and open an X11 window inside your program, the following steps are typically executed:

1. Do an `XOpenDisplay`. This establishes the client/server connection.
2. Use the `XHPGetServerMode` procedure to determine the server operating mode. The server operating modes are discussed in Chapter 4.
3. Do an `XCreateWindow`. This creates a window and returns the window ID.
4. Do an `XMapWindow` to map the window on the display.
5. Call `XFlush` to flush X11's command buffer, validating the window ID.
6. Use the `make_X11_gopen_string` procedure, which takes the window ID and generates a string that can be passed as the ⟨*path*⟩ parameter to the Starbase `gopen` procedure.
7. Do a Starbase open of this window with the `gopen` procedure.
8. Use the procedure `inquire_display_mode` to determine if the X11 server is operating in double-buffer mode. If so, the program can make Starbase double-buffer calls.

The program segment prog2.c below shows this sequence:

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

main()
{
    int fildes;
      :
    if ((Xdisplay = XOpenDisplay(NULL)) == NULL) {
        fprintf(stderr, "%s: can't open %s\n", argv[0], XDisplayName(NULL));
        exit(-1);
    }
    window = XCreateWindow(Xdisplay, ... );
    XFlush(Xdisplay);
    XMapWindow( ... );
    XNextEvent( ... );
    device = make_X11_gopen_string(Xdisplay, window);
    fildes = gopen(device, OUTDEV, "hp98550", INIT);
    inquire_display_mode( ... );
      :
    gclose(fildes);
}
```

This program is linked as follows:

```
cc -o prog2 prog2.o -ldd98550 -lXwindow -lsb1 -lsb2 -lX11hp -lX11
```

To execute this program, type:

```
prog2 Return
```

Note that no parameters need to be passed into the program since the window is created within the program.

Additional Guidelines for Window-Smart Program Development

This section provides additional guidelines for window-smart Starbase program development.

- Refer to the following chapters which provide details on the interaction of Xlib and Starbase:
    - The chapter on Graphics Output (Chapter 4).
    - The chapter on Raster Text (Chapter 5).
    - The chapter on Graphics Input (Chapter 6).
- Window-smart applications that currently run in HP Windows/9000 windows should convert their HP Windows/9000 calls to equivalent Xlib calls. Refer to the document *HP Windows/9000 to X Window System Conversion Guide* for conversion guidelines.
- A window created within a program by `XCreateWindow` will cease to exist when the creating program terminates. This is in contrast to a window created by `xwcreate`, which continues to exist after a program which uses the window terminates. You should ensure that the program which creates the window continues to run if you want the window to continue to exist.

A Hybrid Approach: A Window-Smart Program That Defaults to Raw Mode Operation

You can develop a program that operates as a window-smart program if an X11 server is available and defaults to raw mode operation if an X11 server is not available. This means that your program should first attempt to determine whether an X11 server is available. If the server is available, your program would operate in window-smart mode. If the server is not available, the program operates in raw mode.

For example, a program can attempt to make connection to the X server with the `XOpenDisplay` procedure. If this succeeds, the program can operate as a window-smart program. If this connection fails, the program can operate to the raw display in raw mode.

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 `X`*n*`screens` 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 Supported 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.	All planes supported for Xlib, 8 planes or less supported for Starbase.

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

The HP 98720 display supports up to eight colors in the overlay planes. In the overlay planes, only black, white, and the six primary and secondary colors are supported. The overlay planes also support a transparency color. When the transparency color is written into the overlay planes, the observed color is that of the image planes. Like any color, the transparency color must be written into the color map, and takes up one of the eight colors. Refer to the `gescape` section of the *Starbase Device Drivers Library Manual* for information on how to set the transparency color.

HP 98730 Display

There are either three or four overlay planes in the HP 98730 display. The device file used decides the number of planes available. If only three overlay planes are used, the fourth plane can be used for cursors by Starbase applications running in the image planes. When using a Combined Mode server, even though these planes can display 16 colors simultaneously (8, when using three overlay planes), only 15 (or 7) are available because one color is reserved for the transparency color. This color is always index 15 (or 7). When the transparency color 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.

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 98731 Device Driver

The number of windows that can be opened with the hp98731 driver (which drives the HP 98732A accelerator) 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.

Note that accelerated overlay windows are not supported with the HP 98731 driver.

INIT Absent

The following diagram shows the three phases of interest:

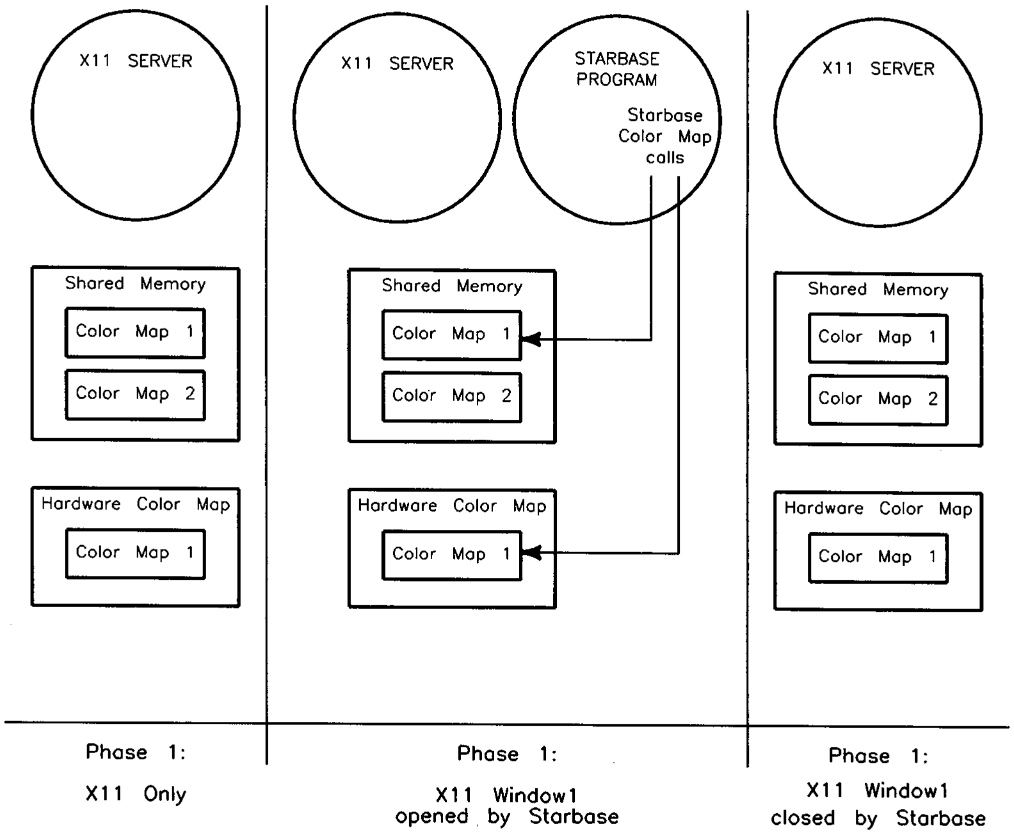

Figure 4–2. Color Map Control with `INIT` Absent

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 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.

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 two cases involving obscured windows and backing store that should be considered:

- 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.

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.

The `hp98731` driver only supports echoes in the cursor plane. When an X11 server is using all four overlay planes, no echoes are supported by the `hp98731` driver. Also, the `hp98731` driver will not pick up echoes created by the `hp98730` driver if the echo is in the image planes.

One exception to this is the Combined Mode server. When a Combined Mode server is using all four overlay planes, both the `hp98730` and `hp98731` drivers place vector echoes in the overlay planes. The echoes use the `WhitePixel` value for the echo color so they should always appear white. If an overlay-plane color map changes the color of `WhitePixel`, the cursors will also change color. Neither the `hp98730` nor the `hp98731` driver place raster cursors in the overlay planes of a four-overlay-plane Combined Mode server.

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. If a separate cursor plane is available, a raw-more Starbase open may place echoes there even if X11 is using the overlay planes.

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.	

X11 Cursor

Always uses the hardware cursor.

HP 98730

Overlay Mode. Echo shares three or four overlay planes if overlay-plane X11 window is opened. Raster echo in image planes if image planes opened. Vector echo in cursor plane if image planes opened, and X11 using three overlay planes. Vector echo in image planes if image planes opened and X11 is using four overlay planes.

Image Mode. Raster echo in image planes if image-plane X11 window opened. Vector echo in cursor plane if image-plane X11 window opened.

Stacked Screen Mode. Echo shares three or four overlay planes if image-plane X11 window is opened. Raster echo in image planes if image-plane X11 window opened. Vector echo in cursor plane if image planes opened, and X11 using three overlay planes. Vector echo in image planes if image planes opened and X11 is using four overlay planes.

Combined Mode. Echo shares three or four overlay planes if overlay-plane X11 window opened. Raster echo in image planes if image-plane X11 window opened. Vector echo in cursor plane if image-plane X11 window opened, and X11 is using three overlay planes. Vector echo in overlay planes if image-plane X11 window opened and X11 is using four overlay planes.

HP 98731

The hp98731 driver cannot open an X11 overlay-plane window.

Overlay Mode. Echo in cursor plane if image planes opened and X11 using three overlay planes. Echo not supported if image planes opened and X11 using four overlay planes.

Image Mode. Echo in cursor plane if image-plane X11 window opened.

Stacked Screen Mode. Echo in cursor plane if image planes opened and X11 using three overlay planes. Echo not supported if image planes opened and X11 using four overlay planes.

Combined Mode. Echo in cursor plane if image-plane X11 window opened, and X11 is using three overlay planes. Vector echo in overlay planes if image-plane X11 window opened and X11 is using four overlay planes. Raster echo is not supported if image-plane X11 window is opened and X11 is using four overlay planes.

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.

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( ... )
    ⋮
```

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 is a *cursor tracking* policy; the window which contains the X11 cursor receives the keyboard input. Some window managers implement an *explicit focus* policy similar to HP Windows/9000; the user explicitly selects a window to receive keyboard input. By default, `hpwm` implements provides an *explicit focus* policy, but can be configured to use a cursor-tracking policy instead. The `uwm` window manager does not implement a focus policy, and thus the default policy—cursor tracking—is in effect. Refer to the documentation for your window manager to understand what type of focus policy, if any, it implements.

Window Managers and Pointer Buttons

One problem that people may see when running Starbase programs that receive input from the X11 pointer device, is that the window manager being run may actually prevent Starbase from seeing the button presses. For example, `hpwm` implements its explicit-focus policy by grabbing all button presses that occur while the X cursor is over a window. Therefore, if a user moves the pointer over a Starbase window and presses the left button, the Starbase program will not see the button press, because `hpwm` has "grabbed" the button press. Typically, the default configuration of window managers is to grab all button presses. However, most window managers allow users to reconfigure the window manager's button bindings in order to allow programs to receive input from the X11 pointer device. Refer to the documentation for your window manager to understand how to do this.

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`

Series 800 systems can have more than one HP-HIL loop and support multiple *seats*. Up to four people can run independent X11 servers. By default, if the display number of the server is 0–3, the server will use the same number of HP-HIL loop. For example, if `DISPLAY` is set to `unix:2.0`, the server will use the HP-HIL loop corresponding to the device files `/dev/hil_2.0` through `/dev/hil_2.6`. These are the same devices that are available to Starbase programs running in a window on that server. A system administrator can easily change this default by creating a `usr/lib/X11/X*devices` file. Refer to the "System-level Customization" chapter of *Using the X Window System, Version 11*. Additional information for a particular release of HP-UX may be found in the `/etc/newconfig/Update_info` directory.

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 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. Additional guidelines pertinent to a particular release of HP-UX may be found in the directory `/etc/newconfig/Update_info` in a file whose name indicates the related release; e.g., `to.6.5` for HP-UX 6.5.

- 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.
- When Starbase is running in an X11 window, it opens connections to the X11 server. Sometimes the X11 server generates events that it sends to all programs that have opened connections to it. These events are sent regardless of whether the programs have any interest in those events. If the server sends many of these events, the connecting socket could overflow. In this case, the X11 server will hang for about 10 seconds, and then close its connection with the program. In order to avoid this happening to programs that run for long periods of time, Starbase programs should occasionally call `make_picture_current`. This procedure causes unwanted events to be cleaned out of the connection with the X11 server.

  On the other hand, *extraneous* calls to `make_picture_current` 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 separately 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.