

Graphics Administration Guide

HP 9000 Servers



HP Part No. B2355-90122
Printed in USA E0497

Edition 1

Notices

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

Hewlett-Packard provides the following material “as is” and 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 (including lost profits) in connection with the furnishing, performance, or use of this material whether based on warranty, contract, or other legal theory.

Some states do not allow the exclusion of implied warranties or the limitation or exclusion of liability for incidental or consequential damages, so the above limitation and exclusions may not apply to you. This warranty gives you specific legal rights, and you may also have other rights which vary from state to state.

Hewlett-Packard assumes no responsibility for the use or reliability of its software on equipment that is not furnished by Hewlett-Packard.

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.

Copyright © 1997 Hewlett-Packard Company This document contains information which is protected by copyright. All rights are reserved. Reproduction, adaptation, or translation without prior written permission is prohibited, except as allowed under the copyright laws.

Restricted Rights Legend. Use, duplication or disclosure by the U.S. Government is subject to restrictions as set forth in subparagraph (c)(1)(ii) of the Rights in Technical Data and Computer Software clause in DFARS 252.227-7013. Rights for non-DoD U.S. Government Departments and Agencies are as set forth in FAR 52.227-19(c)(1,2).

Use of this manual and flexible disc(s), or tape cartridge(s), or CD-ROM supplied for this pack is restricted to this product only. Additional copies of the programs

can be made for security and back-up purposes only. Resale of the programs in their present form or with alterations, is expressly prohibited.

PEX and PEXlib are trademarks of Massachusetts Institute of Technology.

Printing History

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

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

April 1997 ... Edition 1. This manual is valid for HP-UX release 10.30 on all HP 9000 servers.

Preface

Why This Document?

This document was created to fill a need that became evident as Hewlett-Packard began to offer multiple Application Programmer Interfaces (APIs). The situation was this: As HP created one API—say, Starbase—particular aspects of graphical operation were noted and diligently explained in the Starbase documentation. However, some of these aspects were not unique to Starbase, they pertained to graphics operation in general: they applied to *all* of our APIs. Therefore, those users who had HP-PHIGS would be encountering some of the same graphical questions that were already well-documented in the Starbase documentation. But HP-PHIGS users wouldn't necessarily *have* the Starbase documentation. Are they just out of luck? The same situation occurred with HP PEX.

Our dilemma was this: do we copy the general-graphics explanations that already existed in the Starbase documentation, into the documentation for the other APIs as well? This would mean two, three, or even more virtually identical copies of the same explanations in different places, requiring similar changes in each whenever new capabilities or devices were introduced. And if all documents containing these similar explanations were not reprinted simultaneously, “current” documents for the various APIs might contradict each other.

A more elegant solution is: this document. While the API-specific documents still contain most of their previous contents, the general graphical information—common to all APIs—was moved here. Examples include:

- **Pathnames:** File locations have changed between HP-UX 9.x and HP-UX 10.x, with its new, standardized V.4 file system structure.
- **Creating device files:** Regardless of whether it is Starbase, HP-PHIGS, or HP PEX that creates an image, you have to tell the operating system where the display is and how to talk to it.
- **Compiling and Linking:** The process of turning your source code into executable code has many common ideas, regardless of API or file system structure.
- **X Windows issues:** All APIs interact with X windows, so non-unique X Windows information comes here.

The above topics, and others as well, are good candidates for a common area. With this approach, only one copy of the common information need exist, and revisions can happen in a more timely manner, and at less risk of contradicting other documents.

Document Conventions

Below is a list of the typographical conventions used in this document:

<code>mknod /usr/include</code>	Verbatim computer literals are in computer font. Text in this style is letter-for-letter verbatim and, depending on the context, should be typed in exactly as specified, or is named exactly as specified.
In <i>every</i> case ...	Emphasized words are in italic type.
... device is a freem ...	New terms being introduced are in bold-faced type.
... the <i><device_id></i> ...	Conceptual values are in italic type, enclosed in angle brackets. These items are not verbatim values, but are descriptors of the <i>type</i> of item it is, and the user should replace the conceptual item with whatever value is appropriate for the context.

0-2 Preface

Contents

0. Preface	
Why This Document?	0-1
Document Conventions	0-2
1. Pathnames	
Using “whence”	1-1
Using “find”	1-2
Starbase, HP-UX 9.x File System	1-3
Starbase, HP-UX 10.x File System	1-4
HP-PHIGS, HP-UX 9.x File System	1-5
HP-PHIGS, HP-UX 10.x File System	1-6
HP PEX, HP-UX 9.x File System	1-7
HP PEX, HP-UX 10.x File System	1-8
2. Compiling Your Application	
Compiling Starbase Applications	2-2
Compiling with Shared Libraries	2-2
Examples	2-2
Compiling with Archive Libraries	2-3
Examples	2-3
Compiling HP-PHIGS Applications	2-5
Compiling with Shared Libraries	2-5
Compiling with Archive Libraries	2-5
Device Driver Libraries	2-7
Compiling HP PEX Applications	2-8

3. X Windows, HP-UX 9.0x	
The X*screens File	3-1
Miscellaneous Topics	3-2
Setting/Unsetting Environment Variables	3-2
Double Buffer Extension (DBE)	3-3
Performing Buffer Swaps on Vertical Blank	3-3
Supported Devices	3-4
MBX	3-4
Single Logical Screen (SLS)	3-6
Supported SLS Configurations	3-6
HP VUE and Single Logical Screen	3-7
HP Color Recovery	3-7
Dynamic Loading	3-9
Shared Memory Usage	3-9
Changing Graphics Shared Memory Size	3-10
Count Transparent In Overlay Visual	3-10
Enable Overlay Transparency	3-11
3-Bit Center Color	3-11
Image Text Via BitMap	3-11
Obsolete Environment Variables	3-12
Special Device Files	3-13
Supported X Configurations	3-16
Supported Graphics Devices	3-16
Multi-Display Support	3-17
Multi-Screen Support	3-18
Integrated Color Graphics Device-Dependent Information	3-21
Supported Visuals	3-21
Supported Environment Variables	3-21
Colormaps and Colormap Management	3-21
The Default Colormap Management Scheme	3-21
Accessing HP Color Recovery Technology via Xlib	3-22
Internal Color Graphics, Internal GrayScale Graphics, CRX, GRX, andDual-CRX Device-Dependent Information	3-25
Supported Visuals	3-25
Supported Environment Variables	3-25
CRX-24[Z] Device-Dependent Information	3-26
Supported Visuals	3-26
Supported Environment Variables	3-26

CRX-24[Z] Transparent Overlay Visuals	3-26
CRX-48Z Device-Dependent Information	3-28
Supported Visuals	3-28
Supported Environment Variables	3-28
CRX-48Z Transparent Overlay Visuals	3-28
HCRX Device-Dependent Information	3-30
Supported Visuals	3-31
Supported Environment Variables	3-32
HCRX Configuration Hints	3-32
HCRX-8[Z] and HP VISUALIZE-8 Visuals and Double-Buffer Support	3-32
Implications and Suggestions for HCRX-8[Z] and HP VISUALIZE-8	3-33
HCRX Overlay Visuals and Overlay Transparency	3-33
Overlay Transparency on the HCRX-8[Z] and HP VISUALIZE-8	3-33
Overlay Transparency on the HCRX-24[Z], HP VISUALIZE-24, and -48	3-34
HCRX Colormaps	3-36
HCRX-8[Z] and HP VISUALIZE-8: Eight Overlay Planes and Two Depth-8 Banks of Image Planes	3-36
HCRX-24[Z] and HP VISUALIZE-24: 8 Overlay Planes and 24 Image Planes	3-36
HP VISUALIZE-48: Eight Overlay Planes and 48 Image Planes	3-37
Accessing HP Color Recovery Technology via Xlib	3-37
Freedom Series Graphics (S3150, S3250 and S3400)	
Device-Dependent Information	3-41
Supported Visuals	3-41
Supported Environment Variables	3-41
GLX	3-41
VRX Device-Dependent Information	3-42
Supported Visuals	3-42
VRX Device Files	3-42

4. X Windows: HP-UX 10.x	
The X*screens File	4-1
Description of the X*screens Configuration File	4-1
Syntax Guidelines	4-2
The X*screens File Format	4-3
Server Options	4-4
Screen Entries	4-5
Sample X*screens Files	4-7
Miscellaneous Topics	4-14
Double Buffer Extension (DBE)	4-14
Performing Buffer Swaps On Vertical Blank	4-14
Determining Swap Performance	4-15
Supported Devices	4-15
Display Power Management Signaling (DPMS)	4-15
Prior to HP-UX 10.30	4-16
HP-UX 10.30 and Beyond	4-17
MBX	4-18
Shared Memory Extension (MIT_SHM)	4-19
Supported Devices	4-20
Shared Memory Transport (SMT)	4-20
Performance Tuning of SMT	4-21
HP Color Recovery	4-24
HP Color Recovery Extension	4-25
Accessing HP Color Recovery Technology via the Color Recovery Extension	4-26
Dynamic Loading	4-27
Include Inferiors Fix	4-28
Shared Memory Usage With 3D Graphics	4-29
Changing Graphics Shared Memory Size	4-29
Count Transparent In Overlay Visual	4-30
Enable Overlay Transparency	4-30
3-Bit Center Color	4-31
Image Text Via BitMap	4-31
Obsolete Environment Variables	4-32
Special Device Files	4-32
Supported X Configurations	4-35
Supported Graphics Devices	4-35
Multi-Display Support	4-37

Multi-Screen Support	4-40
Single Logical Screen (SLS)	4-43
Supported SLS Configurations	4-44
3D Acceleration and Single Logical Screen	4-44
HP VUE/CDE and Single Logical Screen	4-44
Integrated Color Graphics Device-Dependent Information	4-45
Supported Visuals	4-45
Supported Screen Options	4-46
Colormaps and Colormap Management	4-46
Default Colormap Management Scheme	4-46
Accessing HP Color Recovery Technology via Xlib	4-47
Internal Color Graphics, Internal Grayscale Graphics, CRX, GRX, and Dual-CRX Device-Dependent Information	4-50
Supported Visuals	4-50
Supported Screen Options	4-50
CRX-24[Z] Device-Dependent Information	4-51
Supported Visuals	4-51
Supported Screen Options	4-51
CRX-24[Z] Transparent Overlay Visuals	4-52
CRX-48Z Device-Dependent Information	4-53
Supported Visuals	4-53
Screen Options	4-53
CRX-48Z Transparent Overlay Visuals	4-53
HCRX and HP VISUALIZE Device-Dependent Information	4-54
Supported Visuals	4-55
Supported Screen Options	4-57
HP VISUALIZE-EG Modes	4-57
HCRX Configuration Hints	4-58
HCRX-8[Z], HP VISUALIZE-EG(D) and HP VISUALIZE-8 Visuals and Double-Buffer Support	4-58
Implications and Suggestions for HCRX-8[Z], HP VISUALIZE-EG(D) and HP VISUALIZE-8	4-58
HCRX Overlay Visuals and Overlay Transparency	4-59
Overlay Transparency on the HCRX-8[Z], HP VISUALIZE-EG(D) and HP VISUALIZE-8	4-59
Overlay Transparency on the HCRX-24[Z], HP VISUALIZE-24, and HP VISUALIZE-48	4-60
HCRX Colormaps	4-62

HP VISUALIZE-EG(8): 8 Image planes	4-62
HCRX-8[Z], HP VISUALIZE-EG(D) and HP VISUALIZE-8: Eight Overlay Planes and Two Depth-8 Banks of Image Planes	4-62
HCRX-24[Z] and HP VISUALIZE-24: Eight Overlay Planes and 24 Image Planes	4-63
HP VISUALIZE-48: Eight Overlay Planes and 48 Image Planes	4-63
Accessing HP Color Recovery Technology via Xlib	4-63
Freedom Series Graphics Device-Dependent Information	4-66
Supported Visuals	4-66
Supported Screen Options	4-67
Freedom Video Formats	4-67
VRX Device-Dependent Information	4-68
Supported Visuals	4-68
VRX Device Files	4-69

5. X Windows Configuration Details

Making an X*.hosts File	5-1
XO.hosts and XOscreens Relation	5-2
Using an /etc/hosts File	5-2
Using Special Input Devices	5-3
How the X Server Chooses the Default Keyboard and Pointer X*devices File	5-3
Explicitly Specifying Input Device Use	5-5
Explicitly Specifying RS-232 Input Device Use	5-5
Specifying HP-HIL Input Device Use by Device Type and Position	5-7
Selecting Values for X*devices Files	5-8
Examples	5-9
Specifying HP-HIL Input Device Use by Device File Name	5-10
Redefining the HP-HIL Search Path	5-10
Stopping the X Window System	5-11
Initializing the Colormap with xinitcolormap	5-12
Customizing the Mouse and Keyboard	5-13
Changing Mouse Button Actions	5-13
Going Mouseless with the X*pointerkeys File	5-15
Configuring X*devices for Mouseless Operation	5-15

The Default Values for the <code>X*pointerkeys</code> File	5-16
Creating a Custom <code>X*pointerkeys</code> File	5-16
Syntax	5-16
Assigning Mouse Functions to Keyboard Keys	5-17
Modifier Keys	5-21
Specifying Pointer Keys	5-22
Examples	5-22
Customizing Keyboard Input	5-24
Modifying Modifier Key Bindings with <code>xmodmap</code>	5-24
Specifying Key Remapping Expressions	5-25
Examples	5-27
Printing a Key Map	5-28
Using the Keyboards	5-29
Understanding the Keyboards	5-29
Default Keyboard Mapping	5-30
Equivalent Keys	5-31
Changing Key Mapping	5-32
C1429 Keyboard	5-32
46021 Keyboard	5-32
Comparing the Keyboards	5-33
6. PowerShade: Enhanced 3D Rendering in Software	
Compatibility Considerations	6-1
Re-Installing PowerShade	6-2
HP Series 700 Graphics Performance	6-3
7. Miscellaneous Topics	
3D Thread-Safing	7-1
General Information	7-1
Other Threads-Related Information	7-2
SIGALRM Details	7-2
SIGCHLD and the GRM Daemon	7-3
SIGCHLD and the Starbase Input Daemon	7-4
SIGPIPE Details	7-4
HP CDE and HP VUE	7-5
Shared Memory Usage	7-6
Reference Documentation	7-7

A. Reference
X Windows A-1
X Server A-26

Index

Figures

4-1. Results of Minimal Legal X*screens File	4-7
4-2. Two Physical Displays, Two Separate Screens	4-7
4-3. PVRX/TVRX Display with Overlays	4-9
4-4. Two Physical Displays, Single Logical Screen (1x2)	4-10
4-5. Four Physical Displays, Single Logical Screen (1x4)	4-11
4-6. Four Physical Displays, Single Logical Screen (4x1)	4-12
4-7. Four Physical Displays, Single Logical Screen (2x2)	4-13
4-8. Three Physical Displays, Screen plus Single Logical Screen (1x2)	4-13
4-9.	4-37
4-10.	4-38
4-11.	4-39
5-1. Keycap, Keycode, and Keysym Relationships	5-29

Tables

1-1.	1-3
1-2.	1-4
1-3.	1-5
1-4.	1-6
1-5.	1-7
1-6.	1-8
2-1. Device/Device Driver Correlation	2-7
3-1. Setting and Unsetting Environment Variables	3-2
3-2. Special Device Files: Major/Minor Numbers	3-14
3-3. Graphics Devices Supported on HP-UX 9.x	3-16
3-4. <code>mknod</code> Information for HP-UX 9.x	3-42
4-1. Power-Saving States Defined by VESA	4-16
4-2. Special Device Files on HP-UX 10.x	4-33
4-3. HP 9000 Supported Graphics Devices Table	4-35
4-4. Alternative Supported Freedom Video Formats	4-67
4-5. Alternative Unsupported Freedom Video Formats	4-68
4-6. VRX Device Files	4-69
5-1. Values for <code>X*devices</code> Files	5-8
5-2. Default Mouse Button Mapping	5-13
5-3. Alternative Mouse Button Mappings	5-14
5-4. Pointer Movement Functions	5-17
5-5. Pointer Distance Functions	5-18
5-6. Button Operation Functions	5-18
5-7. Button Mapping Functions	5-19
5-8. Reset and Threshold Functions	5-19
5-9. Button Chording	5-20
5-10. Specifying a Portion of a Tablet	5-21
5-11. Valid <code>xmodmap</code> Expressions	5-26
5-12.	5-31
6-1. Optimized vs. Normal 3D Performance	6-4

Contents-10

Pathnames

This chapter contains information on locating files that reside at some location—currently unknown to you—in the file system. This is important, because you may have an “old” file system (on HP-UX 9.x and earlier) or a “new” V.4 file system (on HP-UX 10.x and later). Most files in the old file system still exist in the new, but they may reside in different locations. This can cause inconvenience if the new location of a file is unknown to you. This chapter addresses the task of finding files, regardless of your file system.

Using “whence”

There are two main methods of finding files, assuming you know the name of the file you’re looking for. The first method is to use the Korn-shell command **whence**, which tells you where commands reside (if you’re not using the Korn shell, you can use the system command **whereis**):

```
$ whence mknod Return
/etc/mknod
```

The above approach, while satisfactory in many cases, has two limitations:

- First, the directory in which the command resides must be one of the entries in the **PATH** variable; if it is not, it won’t be found. So in a sense, **whence** and **whereis** can only find things if you tell them where to look. They are still valuable, though: you may not remember *which*, of the dozens of directories that may be in your **PATH** variable, is where a particular command resides. Also, if you have two commands of the same name in two different directories, **whence** and **whereis** will tell you which one will be found first, and thus executed.
- Secondly, both **whence** and **whereis** only find *executable* files; that is, commands (both compiled programs and shell scripts). If you want to find a file that is *not* executable—an **include** file, for instance—**whence** and **whereis**

will not find it, even if the `include` file's directory is in your `PATH`. To find non-executable files, you can use `find`, discussed below.

Using “find”

The `find` command will find any file in your file system, executable or not. For example, to locate the `include` file we couldn't locate above, you could say:

```
$ find / -name <file_name> Return
```

where `<file_name>` is the name of the file you're looking for. In the above example, the “/” is the root directory, and *everything* is under that, so—assuming you specified the correct file name, and it *is* somewhere in the file system—the above command is guaranteed to find what you're looking for, though it may take a while. You can shorten the search time by giving a subdirectory here, if you know it; for example, “`find /opt ...`”. Also, you can specify just a *partial* filename; `find` will locate all files containing a specified substring in their names. The `find` command has many other options for refining a search; see the reference page for details.

Subsequent sections of this chapter contain the actual pathnames referred to in other HP graphics API documents, such as Starbase, PEX, etc. A particular paragraph might refer to, say, the `< demos >` directory. That directory on an *9.x* system may be in a different location than on a *10.x* system, so the sections below allow you to resolve the actual path name, given the HP-UX operating system version you have, and the API you are working with.

Find the API you're looking for, then under that, the operating system you have (the old *9.x* or the new *10.x* file system). In that section is an alphabetical list of “generic names”—the file system path references used in the other documents—and with each, you will see its actual location in the file system.

1-2 Pathnames

Starbase, HP-UX 9.x File System

Table 1-1.

<i><common></i>	(didn't exist in 9.x file system)
<i><dev></i>	/dev
<i><formatters></i>	/usr/lib/starbase/formatters
<i><nls></i>	/usr/lib/nls
<i><sb-demos></i>	/usr/lib/starbase/demo
<i><sb-fonts></i>	/usr/lib/starbase/stroke
<i><sb-font-info></i>	/usr/lib/starbase/stroke/font_info
<i><sb-incl></i>	/usr/include
<i><sb-lib></i>	/usr/lib
<i><sb-utils></i>	/usr/lib/starbase/demos/SBUTILS
<i><screen></i>	/dev/screen
<i><starbase></i>	/usr/lib/starbase
<i><tmp></i>	/tmp
<i><vue-config></i>	/usr/vue/config
<i><x11></i>	/usr/lib/X11
<i><x11-admin></i>	/usr/lib
<i><x11r5></i>	/usr/lib/X11R5
<i><x11r5-incl></i>	/usr/include/X11R5
<i><xconfig></i>	/usr/lib/X11

Starbase, HP-UX 10.x File System

Note that before HP-UX 10.20, the default release number of the X window system was X11R5; as of HP-UX 10.20, it is X11R6. If your version of HP-UX is 10.20 or newer, the occurrences of X11R5 in the pathnames below should be understood as X11R6.

Table 1-2.

<code><common></code>	<code>/opt/graphics/common</code>
<code><dev></code>	<code>/dev</code>
<code><formatters></code>	<code>/opt/graphics/starbase/formatters</code>
<code><nls></code>	<code>/opt/graphics/common/lib/nls/msg/C</code>
<code><sb-demos></code>	<code>/opt/graphics/starbase/demo</code>
<code><sb-font></code>	<code>/opt/graphics/common/stroke</code>
<code><sb-font-info></code>	<code>/opt/graphics/common/stroke/font_info</code>
<code><sb-incl></code>	<code>/opt/graphics/starbase/include</code>
<code><sb-lib></code>	<code>/opt/graphics/common/lib</code>
<code><sb-utils></code>	<code>/opt/graphics/starbase/demos/starbase/SBUTILS</code>
<code><screen></code>	<code>/dev/screen</code>
<code><starbase></code>	<code>/opt/graphics/starbase</code>
<code><tmp></code>	<code>/var/tmp</code>
<code><vue-config></code>	<code>/etc/vue/config</code>
<code><x11></code>	<code>/usr/lib/X11</code>
<code><x11-admin></code>	<code>/etc/X11</code>
<code><x11r5></code>	<code>/usr/lib/X11R5</code>
<code><x11r5-incl></code>	<code>/usr/include/X11R5</code>
<code><xconfig></code>	<code>/etc/X11</code>

1-4 Pathnames

HP-PHIGS, HP-UX 9.x File System

Table 1-3.

<i><app-defaults></i>	<code>/usr/lib/X11/app-defaults</code>
<i><common></i>	(didn't exist in 9.x file system)
<i><dev></i>	<code>/dev</code>
<i><nls></i>	<code>/usr/lib/nls</code>
<i><phigs></i>	<code>/usr/lib/phigs</code>
<i><phigs-demos></i>	<code>/usr/lib/phigs/demos</code>
<i><phigs-examples></i>	<code>/usr/lib/phigs/examples</code>
<i><phigs-incl></i>	<code>/usr/include</code>
<i><phigs-lib></i>	<code>/usr/lib</code>
<i><phigs-widget></i>	<code>/usr/include/Motif1.2</code>
<i><screen></i>	<code>/dev/screen</code>
<i><spool></i>	<code>/usr/spool</code>
<i><starbase></i>	<code>/usr/lib/starbase</code>
<i><vue-config></i>	<code>/usr/vue/config</code>
<i><x11></i>	<code>/usr/lib/X11</code>
<i><x11r5></i>	<code>/usr/lib/X11R5</code>
<i><x11r5-incl></i>	<code>/usr/include/X11R5</code>
<i><xconfig></i>	<code>/usr/lib/X11</code>

HP-PHIGS, HP-UX 10.x File System

Note that before HP-UX 10.20, the default release number of the X window system was X11R5; as of HP-UX 10.20, it is X11R6. If your version of HP-UX is 10.20 or newer, the occurrences of X11R5 in the pathnames below should be understood as X11R6.

Table 1-4.

<i><app-defaults></i>	<i>/usr/lib/X11/app-defaults</i>
<i><common></i>	<i>/opt/graphics/common</i>
<i><dev></i>	<i>/dev</i>
<i><nls></i>	<i>/opt/graphics/common/lib/nls/msg/C</i>
<i><phigs></i>	<i>/opt/graphics/phigs</i>
<i><phigs-demos></i>	<i>/opt/graphics/phigs/demos</i>
<i><phigs-examples></i>	<i>/opt/graphics/phigs/examples</i>
<i><phigs-incl></i>	<i>/opt/graphics/phigs/include</i>
<i><phigs-lib></i>	<i>/opt/graphics/phigs/lib</i>
<i><phigs-widget></i>	<i>/opt/graphics/phigs/include/Motif1.2</i>
<i><screen></i>	<i>/dev/screen</i>
<i><spool></i>	<i>/var/spool</i>
<i><starbase></i>	<i>/opt/graphics/starbase</i>
<i><vue-config></i>	<i>/etc/vue/config</i>
<i><x11></i>	<i>/usr/lib/X11</i>
<i><x11r5></i>	<i>/usr/lib/X11R5</i>
<i><x11r5-incl></i>	<i>/usr/include/X11R5</i>
<i><xconfig></i>	<i>/etc/X11</i>

1-6 Pathnames

HP PEX, HP-UX 9.x File System

Table 1-5.

<i><app-defaults></i>	<i>/usr/lib/X11/app-defaults</i>
<i><cge-examples></i>	<i>/usr/lib/PEX5/cge_examples</i>
<i><cge-utils></i>	<i>/usr/lib/PEX5/cge_utilities</i>
<i><contrib></i>	<i>/usr/contrib</i>
<i><err-help></i>	<i>/usr/lib/PEX5</i>
<i><extensions></i>	<i>/usr/lib/X11/extensions</i>
<i><hp-examples></i>	<i>/usr/lib/PEX5/hp_examples</i>
<i><man></i>	<i>/usr/man</i>
<i><nls></i>	<i>/usr/lib/nls</i>
<i><ora-examples></i>	<i>/usr/lib/PEX5/ora_examples</i>
<i><peX></i>	<i>/usr/lib/PEX5</i>
<i><peXd></i>	<i>/usr/bin/X11</i>
<i><peX-examples></i>	<i>/usr/lib/PEX5/examples</i>
<i><peX-fonts></i>	<i>/usr/lib/PEX5/fonts</i>
<i><peX-incl></i>	<i>/usr/include</i>
<i><peX-lib></i>	<i>/usr/lib</i>
<i><peX-utils></i>	<i>/usr/lib/PEX5/utilities</i>
<i><profile></i>	<i>/usr/contrib/PEX5/lib</i>
<i><rel-notes></i>	<i>/etc/newconfig</i>
<i><spool></i>	<i>/usr/spool</i>
<i><vhelP></i>	<i>/usr/vhelp</i>
<i><vue></i>	<i>/usr/vue</i>
<i><x11></i>	<i>/usr/lib/X11</i>
<i><x11-incl></i>	<i>/usr/include/X11</i>
<i><x11r5></i>	<i>/usr/lib/X11R5</i>
<i><x11r5-incl></i>	<i>/usr/include/X11R5</i>

HP PEX, HP-UX 10.x File System

Note that before HP-UX 10.20, the default release number of the X window system was X11R5; as of HP-UX 10.20, it is X11R6. If your version of HP-UX is 10.20 or newer, the occurrences of X11R5 in the pathnames below should be understood as X11R6.

Table 1-6.

<i><app-defaults></i>	/usr/lib/X11/app-defaults
<i><cge-examples></i>	/opt/graphics/PEX5/examples/cge
<i><cge-utils></i>	/opt/graphics/PEX5/utilities/cge
<i><contrib></i>	/opt/graphics/PEX5/contrib
<i><err-help></i>	/opt/graphics/PEX5/help5.1
<i><extensions></i>	/opt/graphics/PEX5/newconfig/usr/lib/X11/extensions
<i><hp-examples></i>	/opt/graphics/PEX5/examples/hp
<i><man></i>	/opt/graphics/PEX5/share/man
<i><nls></i>	/opt/graphics/PEX5/lib/nls/msg/C
<i><ora-examples></i>	/opt/graphics/PEX5/examples/OReilly
<i><peX></i>	/opt/graphics/PEX5
<i><peXd></i>	/opt/graphics/PEX5/sbin
<i><peX-examples></i>	/opt/graphics/PEX5/examples
<i><peX-fonts></i>	/opt/graphics/PEX5/fonts
<i><peX-incl></i>	/opt/graphics/PEX5/include/X11R5/X11/PEX5
<i><peX-lib></i>	/opt/graphics/PEX5/lib
<i><peX-utils></i>	/opt/graphics/PEX5/utilities
<i><profile></i>	/opt/graphics/PEX5/contrib
<i><rel-notes></i>	/opt/graphics/PEX5/newconfig/opt/graphics/PEX5
<i><spool></i>	/var/spool
<i><vhelP></i>	/opt/graphics/PEX5/help5.1
<i><vue></i>	/usr/vue

1-8 Pathnames

Table 1-6. (continued)

<code><x11></code>	<code>/opt/graphics/PEX5/newconfig/usr/lib/X11</code>
<code><x11-incl></code>	<code>/usr/include/X11R5/X11</code>
<code><x11r5></code>	<code>/opt/graphics/PEX5/lib/X11R5</code>
<code><x11r5-incl></code>	<code>/usr/include/X11R5</code>

Compiling Your Application

This chapter provides information for compiling you application with either archived or shared libraries for the following Application Programming Interfaces (APIs): Starbase, HP-PHIGS, and HP PEX. Compiling examples are given for C, Fortran, and Pascal.

The actual pathnames of the conceptual (*italicized and angle-bracketed*) directory names in this chapter depends on the file system structure, which differs between HP-UX 9.x and HP-UX 10.x. See Chapter 1 for details.

Compiling Starbase Applications

Compiling with Shared Libraries

2 The compiler programs (`cc`, `f77`, and `pc`) link with Starbase shared libraries by default. Starbase will explicitly load the appropriate device driver library at run time when you compile and link with the shared library `<common>/lib/libhpgfx.sl`, or use the `-lhpgfx` option. This loading occurs at `gopen(3G)` time.

Examples

Assuming you are using `ksh(1)`, to compile and link a C program for use with the shared library driver, use the forms below. Again, if your version of the operating system is HP-UX 10.20 or newer, the X11 release number should be X11R6, not X11R5.

```
cc example.c -I/usr/include/X11R5/X11 -I <sb-incl> \
-L<common> -L<x11r5> -L<sb-lib> \
-lXwindow -lhpgfx -lXhp11 -lX11 -lm -o example
```

For FORTRAN:

```
fort77 example.f -I/usr/include/X11R5/X11 -I<sb-incl> \
-L<common> -L<x11r5> -L<sb-lib> \
-lXwindow -lhpgfx -lXhp11 -lX11 -lm -o example
```

For Pascal:

```
pc example.p -I /usr/include/X11R5/X11 -I<sb-incl> \
-Wl,-L<common> -Wl,-L<x11r5> \
-Wl,-L<sb-lib> -lXwindow -lhpgfx -lXhp11 -lX11 -lm -o example
```

2-2 Compiling Your Application

Compiling with Archive Libraries

You can link the appropriate library, for your specific device driver, to a program by using any one of the following:

- The path name `<sb-lib>/<library_name>.a`;
- An appropriate relative path name; or
- The `-ldd<device_driver>` option (for example, `-lddhcrx`) with the `LDOPTS` environment variable set to `-a archive` and exported.

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

As of HP-UX 9.05, archive libraries utilize functionality that is included in `libXext.a`. Because the archive library `libhpgfx1.a` references functionality in `libXext.a`, it is necessary to explicitly link `libXext.a` with your program. Otherwise, the linker will have undefined references.

Examples

Assuming you are using `ksh(1)`, to compile and link a C program for use with this driver, use the forms below. Again, if your version of the operating system is HP-UX 10.20 or newer, the X11 release number should be `X11R6`, not `X11R5`.

The `-l:libdld.sl` below specifies the dynamic loader, which is available only in shared-library form.

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

and then:

```
export CCOPTS="-I <sb-include>/X11R5/X11"
cc example.c -I/usr/include/X11R5/X11 -I<sb-include> \
-L<common> -L<x11r5> -L<sb-lib> \
-ldd<device_driver> -lXwindow -lhpgfx1 -lhpgfx2 -lXhp11 \
-lX11 -lXext -Wl,-E -Wl,+n -l:libdld.sl -lm -o example
```

For FORTRAN, use:

```
fort77 example.f -I/usr/include/X11R5/X11 -I<sb-include> \
-L<common> -L<x11r5> -L<sb-lib> \
-ldd<device_driver> -lXwindow -lhpgfx1 -lhpgfx2 -lXhp11 \
-lX11 -lXext -Wl,-E -Wl,+n -l:libdld.sl -lm -o example
```

For Pascal, use:

```
pc example.p -I/usr/include/X11R5/X11 -I<sb-incl> \
-Wl,-L<common> -Wl,-L<x11r5> \
-Wl,-L<sb-lib> -ldd<device_driver> \
-lXwindow -lhpgfx1 -lhpgfx2 -lXhp11 -lX11 -lXext \
-Wl,-E -Wl,+n -l:libdld.sl -lm -o example
```

2

2-4 Compiling Your Application

Compiling HP-PHIGS Applications

If your version of the operating system is HP-UX 10.20 or newer, the X11 release number should be X11R6, not X11R5.

Compiling with Shared Libraries

If you are using shared libraries, as we recommend, linking is device-independent. To compile a C program using shared libraries, you would use the following command:

```
cc example.c -I<x11r5-incl> -I<phigs-incl> \
-L<common>/lib -L<phigs-lib> \
-I<phigs-widget>/Motif1.2 -L<x11r5> \
-lXwindow -lphigs -ldl -lhpgfx -ldld \
-lXhp11 -lXi -lXext -lX11 -lm -o example
```

FORTRAN users can simply replace `cc` with `fort77` in the above command. Also, if you are a FORTRAN user and prefer using the `f77` command, you can replace `cc` with `f77` and change linking options that are specified as follows:

```
-L<pathname>
to
-Wl,-L<pathname>
```

For more information on compiling and linking, read the section “PHIGS PLUS Differences Between HP-PHIGS 2.2/2.3 and 3.0” in the chapter “Functional Overview” in the *HP-PHIGS Graphics Techniques* manual.

Compiling with Archive Libraries

If you are using archived libraries, you need to include your device’s driver library. Note that shared libraries are used by default unless you specify (with the environment variable `LDOPTS`) that you want to use archived libraries. You can set this environment variable in the POSIX, Korn, or Bourne shell to specify the use of archived libraries by executing the following commands at the shell prompt or placing them in your shell’s start-up script (`.profile`):

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

If you are using the C shell, you can set the `LD_OPTS` environment variable to specify the use of archived libraries by executing the following command at the shell prompt or placing it in your shell's start-up script (`.cshrc`):

```
setenv LD_OPTS "-a archive"
```

To compile a C program using archived libraries, you would use the following command:

```
cc example.c -I<x11r5-include> -I<phigs-include> \
-L<common>/lib -L<phigs-lib> \
-I<phigs-widget>/Motif1.2 -L<x11r5> \
-lddd1<device_drivers> -Wl,-E -Wl,+n -l:libdld.sl -lXwindow \
-lphigs -ldl -lhpgfx1 -lhpgfx2 -lXhp11 -lXi -lXext -lX11 \
-lm -o example
```

The “`-l:libdld.sl`” above specifies the dynamic loader, which is available only in shared-library form.

Multiple graphics device driver libraries may be indicated in the `<device_drivers>` location. For example, if your application source file is called `app_one.c` and the executable is `app_one` and you are using the CRX graphics device driver (`libddgcrx`), your compile command would look like this:

```
cc app_one.c -I<x11r5-include> -I<phigs-include> \
-L<common>/lib -L<phigs-lib> \
-I<phigs-widget>/Motif1.2 -L<x11r5> \
-lddd1 -lddgrx -Wl,-E -Wl,+n -l:libdld.sl -lXwindow -lphigs \
-ldl -lhpgfx1 -lhpgfx2 -lXhp11 -lXi -lXext -lX11 -lm -o app_one
```

The “`-l:libdld.sl`” above specifies the dynamic loader, which is available only in shared-library form.

Fortran users can simply replace `cc` with `fort77` in the above command. Also, if you are a Fortran user and prefer using the `f77` command, you can replace `cc` with `f77` and change linking options that are specified as follows:

```
-L<pathname>
to
-Wl,-L<pathname>
```

For more information on compiling and linking, read the section “PHIGS PLUS Differences Between HP-PHIGS 2.2/2.3 and 3.0” in the chapter “Functional Overview” in the *HP-PHIGS Graphics Techniques* manual.

2-6 Compiling Your Application

Device Driver Libraries

The following table lists the device driver libraries that should be used with your particular device driver if you are using archived libraries.

Table 2-1. Device/Device Driver Correlation

<code>libddgcrx.a:</code>	HP GRX; HP CRX; HP Dual CRX; HP CRX-24; HP CRX-24Z; Integrated Graphics: <ul style="list-style-type: none"> ■ Color (1280×1024), ■ Color (1024×768), and ■ Grayscale (1280×1024); ■ Internal Color Graphics
<code>libddcrx48z.a:</code>	HP CRX-48Z
<code>libddhcrx.a:</code>	HP HCRX-8; HP HCRX-8Z; HP HCRX-24; HP HCRX-24Z
<code>libdd98705.a:</code>	PersonalVRX
<code>libdd98766.a:</code>	TurboVRX (accelerated)
<code>libddhpgl.a:</code>	HP-GL devices with HP-IB interface
<code>libdvio.a:</code>	HP-GL devices with RS-232 interface
<code>libddCADplt.a:</code>	HP HP-GL/2; HP CADplt; HP CADplt2
No additional libraries required:	HP VMX

Compiling HP PEX Applications

HP PEXlib is supported on the Series 700 workstations using shared libraries that must be linked with the application program. Only PEX programs written in C (not FORTRAN or Pascal) are supported.

When you compile your PEXlib programs, you must link the application with the PEXlib library `libPEX5`. Note that the PEX library is dependent on the math library.

A compile line will typically appear:

```
cc program.c -I <include>/X11R5 -L<x11r5> \
-lPEX5 -lXext -lX11 -lm'
```

For more information on compiling and linking PEXlib programs, see the appropriate chapters in the *HP PEX Implementation and Programming Supplement*.

X Windows, HP-UX 9.0*x*

This chapter documents information specific to the HP X server on HP-UX 9.0*x*. It describes features unique to HP's X server, provides information on how to configure the X server and includes a list of supported X configurations. For each supported graphics device, device-dependent configuration information is provided.

Information specific to a new release of the X server, beyond the scope of the general information in this document, can be found in the HP-UX release notes located in `/etc/newconfig`.

If you prefer to see this information in an ASCII file, please refer to the file `/usr/lib/X11/Xserver/info/screens/hp`. It includes the same information as is contained in this chapter.

The X*screens File

The `X*screens` file is used to configure the operation of the X server. Please refer to the sample file in `/usr/lib/X11/X0screens`, for more information on how to use the `X*screens` file.

Miscellaneous Topics

Setting/Unsetting Environment Variables

Each type of shell (`sh`, `csh` and `ksh`) have different ways of setting and unsetting the value of environment variables to desired values. The following table shows an example of how the `DISPLAY` variable is set (where `<value>` would be in the `<host>:<display>.<screen>` format) and unset for each shell type.

Table 3-1. Setting and Unsetting Environment Variables

Shell	Setting the Variable	Unsetting the Variable
<code>sh</code>	<code>DISPLAY=<value></code> <code>export DISPLAY</code>	<code>unset DISPLAY</code>
<code>ksh</code>	<code>export DISPLAY=<value></code>	<code>unset DISPLAY</code>
<code>csh</code>	<code>setenv DISPLAY <value></code>	<code>unsetenv DISPLAY</code>

Throughout this file, when an environment variable is defined, it will include a list of valid values or will state that any value is acceptable. When any value is specified, any non-null string is acceptable, but often the value is set by convention, to `TRUE` (e.g., “`export <variable>=TRUE`” in `ksh`).

Double Buffer Extension (DBE)

DBE is an extension to the X server that provides a double-buffering Application Programming Interface (API). Note that MBX (the Multi-Buffering eXtension to X) has not been adopted as an industry standard, as DBE has. Thus, it is recommended that applications that use MBX be ported to DBE usage in preparation for future MBX obsolescence. For more information about DBE and the API, consult the DBE man pages:

- DBE
- XdbeQueryExtension
- XdbeGetVisualInfo
- XdbeFreeVisualInfo
- XdbeAllocateBackBufferName
- XdbeDeallocateBackBufferName
- XdbeSwapBuffers
- XdbeBeginIdiom
- XdbeEndIdiom
- XdbeGetBackBufferAttributes

Performing Buffer Swaps on Vertical Blank

For performance reasons, the default DBE behavior is to *not* synchronize buffer swaps with the monitor's vertical retrace period. In some instances, therefore, image tearing (seeing part of the old image and part of the new image on the display at the same time) could be visible while swapping large DBE windows. For those instances where tearing would occur and is undesirable, an optional X server mode is available to allow for synchronization of buffer swaps with vertical retrace. To activate this optional X server mode, set the environment variable `SWAP_BUFFERS_ON_VBLANK` to any value before the X server is started.

Supported Devices

The X server supports DBE on the following devices:

- Internal Color Graphics
- Integrated Color Graphics
- Color Graphics cards
- Dual Color Graphics cards
- CRX
- CRX-24[Z]
- CRX-48Z
- HCRX-8[Z]
- HCRX-24[Z]
- HP VISUALIZE-8
- HP VISUALIZE-24
- HP VISUALIZE-48
- Freedom Series™ Graphics: S3150, S3250 and S3400. (“Freedom Series” is a trademark of Evans & Sutherland Computer Corporation.)

MBX

The MBX extension (Multi-Buffering Extension) is supported on all graphics devices supported on the HP 9000/700 machines, except the PersonalVRX and the TurboVRX.

HP’s implementation of MBX exists mainly to support fast double-buffering for PEX applications. Therefore, MBX only supports allocation of one or two MBX buffers; no more. Some graphics devices/visuals have a single 8-plane buffer; these include Internal Color Graphics, Integrated Color Graphics, Color Graphics cards, Dual Color Graphics cards and the overlay planes on the CRX-24[Z], CRX-48Z, the HCRX family and the Freedom Series Graphics (S3150, S3250 and S3400). For these devices, MBX double-buffering is still supported, but the second bank is allocated in virtual memory. Rendering and buffer-swapping in these instances is slower than devices/visuals that support true hardware double-buffering.

Currently, there is no easy way to determine which visuals, from a device’s list of visuals, support fast MBX hardware double-buffering. The CRX and Dual-CRX device is a double-buffered device and therefore always supports MBX hardware double-buffering. The Internal Color Graphics, Integrated Color Graphics, Color

3-4 X Windows, HP-UX 9.0x

Graphics cards and Dual Color Graphics cards only support MBX software buffering. All other devices that have both overlay and image planes support fast MBX hardware double-buffering in the image planes and slower MBX software double-buffering in the overlays. Consult the following device-specific sections for a list of visuals that support software and hardware MBX double-buffering.

For performance reasons, the default MBX behavior is to *not* synchronize with the monitors vertical retrace period. In some instances, image tearing could be visible while swapping large MBX windows. For those instances where tearing would occur and is undesirable, an optional X server mode is available to allow for synchronization with vertical retrace. To activate this optional X server mode, set the environment variable `SWAP_BUFFERS_ON_VBLANK` to any value before the X server is started.

With this mode enabled, all MBX buffer swaps are synchronized with the monitor's vertical retrace period. This mode is not needed in drawables used for PEX rendering. PEX turns synchronization on and thus does not require this tuning.

The MBX Application Programming Interface is thoroughly discussed in the O'Reilly & Associates, Inc. *PEXlib Programming Manual* by Tom Gaskins. Consult that manual to understand the creation, manipulation, and destruction of MBX buffers.

Since MBX is not an industry standard, developers should replace MBX calls with DBE calls.

Note

Note that `XmbufGetScreenInfo()` can indicate that a window supports MBX even if only one MBX buffer is supported. An application should always check the `max_buffers` field in the returned `XmbufBufferInfo` structure before assuming that a window supports two MBX buffers.

Single Logical Screen (SLS)

SLS is a mechanism for treating homogeneous multi-display configurations as a single “logical” screen. This allows the moving/spanning of windows across multiple physical screens. “Homogeneous” means SLS only works if the graphics devices included in the “SLS Configuration” are of the same type (see the list of the supported SLS configurations shown below.) SLS is enabled via the `/usr/lib/X11/XOscreens` file with the syntax:

```
SingleLogicalScreen n m
    /dev/crt0 ... /dev/crtk
```

where n =the number of “rows” in the physical configuration, m =the number of “columns” in the physical configuration, and the product of $n \times m$ is less than or equal to four.

For example, to create a logical screen that is one screen tall by two screens wide, the following syntax would be used:

```
SingleLogicalScreen 1 2
    /dev/crt0 /dev/crt1
```

Whereas for a logical screen that is two screens tall by one screen wide, the syntax would be:

```
SingleLogicalScreen 2 1
    /dev/crt0 /dev/crt1
```

Supported SLS Configurations

- Series 712 with an Integrated Color Graphics + plug-in Color Graphics
- Series 715 with an Integrated Color Graphics + plug-in Color Graphics
- Series 720 with a Dual CRX
- Series 725 with an Integrated Color Graphics + plug-in Color Graphics
- Series 730 with a Dual CRX
- Series 735 with a Dual CRX
- Series 750/755 with a Dual CRX
- Series 750/755 with two Dual CRXs
- Series 750/755 with two CRX24s
- Series 750/755 with two CRX24Zs
- Series 770 with two HCRX24s
- Series 770 with a Duet

3-6 X Windows, HP-UX 9.0x

HP VUE and Single Logical Screen

Please note that HP VUE has not been modified to take advantage of the Single Logical Screen capability. When presenting information on your display, HP VUE may split a window across physical screens. Examples include:

- The login screen.
- The Front Panel.
- Window move and resize boxes.
- The screen lock dialog.

This behavior is the result of HP VUE's naive assumption that it is running against one large screen; it centers these windows accordingly.

If you are using the default HP VUE key bindings, you can easily reposition the Front Panel so that it is completely contained within one physical screen:

1. With the input focus on the Front Panel, press **(Alt)-(Space)** (on older keyboards, use **(Extend Char) (Space)**).
2. With the Front Panel menu posted and the "Move" menu item selected, press **(Enter)** (on older keyboards, **(Return)**) to start the move.
3. Use the mouse or the arrow keys to reposition the Front Panel to the desired location.
4. Press **(Enter)** (or **(Return)**) to complete the move. You may instead press **(Esc)** to cancel the move.

Afterwards, this setting will be remembered and restored at your next login. If you have previously set a Home session, you will need to re-set the Home session in the Style Manager to register the new Front Panel position.

Note that there is no mechanism in HP VUE for repositioning the login screen, window move/resize boxes, or the screen lock dialog.

HP Color Recovery

Color Recovery is a technique that generates a better picture by eliminating the graininess caused by traditional dithering techniques. It is available on these graphics devices:

- Integrated Color Graphics and plug-in Color Graphics cards
- HCRX: HCRX-8[Z], HCRX-24[Z], HP VISUALIZE-8, HP VISUALIZE-24, HP VISUALIZE-48

Color Recovery is available when using either `PseudoColor` or depth-8 `TrueColor` visuals.

There are two components to the Color Recovery process. First, a different dither-cell size (16×2) is used when rendering shaded polygons. Second, a digital filter is used when displaying the contents of the frame buffer to the screen.

Under some conditions, Color Recovery can produce undesirable artifacts in the image (this also happens with dithering, but the artifacts are different). However, images rendered with Color Recovery are seldom worse than what dithering produces. In most cases, Color Recovery produces significantly better pictures than dithering.

Color Recovery is available by default for all depth-8 color visuals on devices that support the feature. If, for some reason, you wish to disable Color Recovery, set the `HP_DISABLE_COLOR_RECOVERY` environment variable to any value before starting the server.

Color Recovery is enabled in conjunction with a particular X colormap that is associated with your window. If the X colormap is not installed in hardware, you may not see the effect of the Color Recovery filter (you may not even see the correct colors for that window). Given that more than one hardware colormap (or “color lookup table”) is available, this should happen infrequently.

The Color Recovery colormap is a *read-only* colormap. Any attempts to change it will be ignored and no error will be reported.

Access to the Color Recovery capability is transparent when using a 3D graphics API such as Starbase, HP-PHIGS or PEX. If you are producing graphics using Xlib calls, your application must perform some of the necessary processing. The method to access Color Recovery via Xlib is described in a section called “Accessing HP Color Recovery Technology via Xlib” in the device-dependent sections.

Dynamic Loading

HP's X server now dynamically loads the appropriate device drivers and extensions based on the target graphics display device and the extensions it supports. This feature should be transparent to X server users.

The extension/device relationships are explained in the extension specific sections of this file.

Note



Altering or removing files under `/usr/lib/X11/Xserver` may prevent the X server from running.

3

Shared Memory Usage

Graphics processes use shared memory to access data pertaining to the display device and X11 resources created by the server. (“Resources” includes windows, colormaps, and cursors.) The X11 server initiates an independent process called the Graphics Resource Manager (GRM) to manage these resources among graphics processes. Graphics processes include PEXlib, PHIGS, and Starbase applications. One problem encountered with GRM shared memory is that it may not be large enough to run some applications.

Graphics applications that require VM double-buffering (for example, when the `HP_VM_DOUBLE_BUFFER` environment variable is used) use large amounts of shared memory. Shared memory can be completely consumed by several double-buffered graphics windows. When an application attempts to use more shared memory than is available, the application encounters errors and might terminate.

You can circumvent the problem by using environment variables to change the shared memory size.

Changing Graphics Shared Memory Size

The size of the shared memory segment used by the GRM can be controlled through an environment variable. The default value is 0x580000 (5.5 Mbytes) on Series 700 computers.

Note



The actual GRM shared memory size on a system can be determined by running “`ipcs -ma`”, finding the entry with CPID matching the process ID of the `grmd` process and then checking the segment size (`SEGSZ`) field.

3

If more shared memory space is needed, graphics shared memory size can be increased. For example, to set it to eight megabytes in `ksh`:

```
export GRM_SIZE=0x800000
```

Note that the value must be in hexadecimal. The new value won't take effect until you restart the X11 server.

It is also possible to decrease the size of GRM shared memory. You may want to do this if you want to reduce the swap space requirements of your system and/or you do not intend to run any 3D graphics processes. For example, you could reduce graphics shared memory size to 0x100000 (one megabyte).

Count Transparent In Overlay Visual

In some configurations, an 8-plane overlay visual may have less than 256 colors. This should not cause a problem for most applications. If an application depends on 8-plane visuals having 256 colormap entries, this option may be useful. Setting this option to any value will cause the X server to count transparent entries in the number of colormap entries.

Examples of Relevant Graphics Devices: CRX-24[Z], CRX-48Z, HCRX-8[Z], HCRX-24[Z], HP VISUALIZE-8, HP VISUALIZE-24, HP VISUALIZE-48

Environment Variable To Use: HP_COUNT_TRANSPARENT_IN_OVERLAY_VISUAL

Enable Overlay Transparency

This option is used to enable the usage of an overlay transparent color on devices that can support it, but, by default, do not allow it (for example, HCRX-8).

Examples of Relevant Graphics Device: HCRX-8[Z], HP VISUALIZE-8

Environment Variable To Use: HP_ENABLE_OVERLAY_TRANSPARENCY

The variable may be set to any value before starting the X server.

Note that setting this variable will cause the number of colormaps to drop to one in the Overlay planes and one in the Image planes. See the section on the HCRX family of devices for more information.

3-Bit Center Color

This option is available to force the X server to center colors in the colormap to values that will reduce the amount of twinkle on flat-panel conversion. This option applies only to flat-panel displays.

The twinkling effect is caused by the analog-to-digital conversion. Due to noise in the analog signal, it is possible for a color near a boundary between two digital values to cause the conversion to bounce back-and-forth between the two colors (i.e., twinkle). In order to avoid this effect, the server “centers” the colors as far from the color boundaries as possible.

Examples of Relevant Graphics Device: Integrated Color Graphics, Color Graphics cards, Internal Color Graphics

Environment Variable To Use: HP_3_BIT_CENTERCOLOR

The variable may be set to any value before starting the X server.

Image Text Via BitMap

When using the Xlib `XDrawImageString()` call to draw text, a visual effect may be seen where text appears to flicker as the background and foreground are drawn in distinct graphics operations. This option is available to eliminate the flicker effect but at the expense of reduced text performance. The option will make the X server first draw text to an off-screen pixmap prior to displaying it to the screen.

Examples of Relevant Graphics Device: Integrated Color Graphics, Color Graphics cards, CRX-24[Z], CRX-48Z, HCRX-8[Z], HCRX-24[Z], HP VISUALIZE-8, HP VISUALIZE-24, HP VISUALIZE-48

Environment Variable To Use: HPGCRX_IMAGETEXT_VIA_BITMAP

The variable may be set to any value before starting the X server.

Note Using this option will reduce text performance.



3

Obsolete Environment Variables

These environment variables are no longer supported:

- HP_SUPPRESS_TRUECOLOR_VISUAL
- HP_COLORMAP_MANAGEMENT_SCHEME

These environment variables are being replaced and may not be supported in future releases:

- Old Name: WMSHMSPC
New Name: GRM_SIZE
- Old Name: MBX_SWAP_BUFFERS_ON_VBLANK
New Name: SWAP_BUFFERS_ON_VBLANK
- Old Name: CRX24_COUNT_TRANSPARENT_IN_OVERLAY_VISUAL
New Name: HP_COUNT_TRANSPARENT_IN_OVERLAY_VISUAL

Special Device Files

Special device files are used to communicate between the computer and peripheral devices. The X server requires the use of a special device file for each graphics card present in the system.

Special device files are created with the `mknod` command. The `mknod` command resides in `/etc` and may only be invoked by a superuser (i.e., `root`). Although special device files can be made in any directory of the HP-UX file system, the convention is to create them in the `/dev` directory. Any name may be used for the special device file; however, the names that are suggested for the devices are `crt`, `crt0`, `crt1`, `crt2`, or `crt3`. It is also acceptable to use a name that is descriptive of the graphics device, for example, `crt1.left` or `crt1.right`. The usage statement for the `mknod` command is:

```
mknod: arg count
usage: mknod name b|c major minor [ cnode ]
       mknod name p
```

All graphics special device files are character device files with read-write permissions by all. For 9.0x systems, the major number will always be 12 (On 10.0 systems, the major number will always be 174). The following table, in which the leading “0x” indicates that the number is in hexadecimal format, indicates which minor numbers to use for creating alternate device files:

Table 3-2. Special Device Files: Major/Minor Numbers

Device Filename	9.0x Minor Number	10.x Minor Number	Description
/dev/crt	0x100000	0x000000	Standard console device file
/dev/crt.r	0x100000	0x000000	Dual CRX Graphics console, right device
/dev/crt.l	0x100004	0x000004	Dual CRX Graphics console, left device
/dev/hcrx	0x000000	0x010000	Secondary graphics device file
/dev/freedom	0x000000	0x010000	Freedom Series, secondary graphics device file
/dev/crt1.r	0x000000	0x010000	Secondary graphics device is Dual CRX Graphics, right device
/dev/crt1.l	0x000004	0x010004	Secondary graphics device is Dual CRX Graphics, left device
/dev/crt2	(note 1)	0x020000	Third graphics device file
/dev/crt3	(note 1)	0x030000	Fourth graphics device file

Following are some examples of using the `mknod` entry for the HP-UX Operating System.

For an SPU with only one SGC interface slot (e.g., a Model 720) running Dual CRX graphics, a sample 9.07 `mknod` entry for the second graphics device would be:

```
/etc/mknod /dev/crt0.left c 12 0x000004
chmod 666 /dev/crt0.left
```

3-14 X Windows, HP-UX 9.0x

For an SPU with two SGC interface slots, a sample mknod entry for the other slot would be:

```
/etc/mknod /dev/crt1.right c 12 0x000000  
chmod 666 /dev/crt1.right
```

Note that once the device file has been created, it is necessary to ensure that it has read-write permissions by all; i.e., “chmod 666”.

Supported X Configurations

Supported Graphics Devices

The table below summarizes the graphics devices that are supported on each of the HP9000 Series systems:

Table 3-3. Graphics Devices Supported on HP-UX 9.x

Graphics Devices	Supported HP 9000 Models
Integrated GrayScale Graphics ¹	712 (all models), 715/64, 715/80, 715/100, 725/100
Integrated Color Graphics ²	712 (all models), 715/64, 715/80, 715/100, 725/100, 748i/64, 748i/100, V743/64, V743/100
Color Graphics card	712 (all models), 715/64, 715/80, 715/100, 725/100, 748i/64, 748i/100, J200, J210
Dual Color Graphics card	715/64, 715/80, 715/100, 725/100, J200, J210
Internal GrayScale Graphics ¹	705, 710, 715/33, 715/50, 715/75, 725/50, 725/75
Internal Color Graphics ²	705, 710, 715/33, 715/50, 715/75, 725/50, 725/75, 745i/50, 745i/100, 747i/50, 747i/100
CRX, Dual CRX	720, 730, 735, 735/125, 750, 755, 755/125, 747i/50, 747i/100
GRX	720, 730, 735, 735/125
CRX-24	715/33, 715/50, 715/75, 720, 725/50, 725/75, 730, 735, 735/125, 747i/50, 747i/100, 750, 755, 755/125
CRX-24Z	715/33, 715/50, 715/75, 720, 725/50, 725/75, 730, 735, 735/125, 750, 755, 755/125

**Table 3-3.
Graphics Devices Supported on HP-UX 9.x (continued)**

Graphics Devices	Supported HP 9000 Models
CRX-48Z	715/50, 715/75, 715/100, 725/50, 725/75, 725/100, 735, 735/125, 755, 755/125, J200, J210
HCRX-8, HCRX-8Z, HCRX-24, HCRX-24Z, HP VISUALIZE-8, HP VISUALIZE-24	715/64, 715/80, 715/100, 725/100, J200, J210
PersonalVRX, TurboVRX	720, 730, 735, 735/125, 750, 755, 755/125 (no longer on the Corporate Price List)
Freedom Series™ Graphics (S3150, S3250 and S3400)	715/80, 715/100, J200, J210
HP VISUALIZE-48	J200, J210

3

- ¹ Integrated GrayScale Graphics and Internal GrayScale Graphics is supported on high-resolution (1280×1024) for all Models specified above.
- ² Integrated Color Graphics and Internal Color Graphics are supported on both medium-resolution (1024×768) and high-resolution (1280×1024) configurations of the Series 700 Models 705, 710, 712 (all models), and 715/33. High resolution is supported on all other Models specified above.

Multi-Display Support

The following definitions are included to help alleviate confusion between the terms multi-display, multi-screen, multi-seat, and single logical screen.

- **Multi-Display:** A configuration with multiple graphics devices (displays) used concurrently. Any multi-seat or multi-screen configuration is referred to as a multi-display configuration.
- **Multi-Screen:** A configuration in which a single X server with a mouse and keyboard drives multiple graphics devices (where each head is a different X11 screen) concurrently while only allowing the cursor, not windows, to be moved between heads.
- **Multi-Seat:** A configuration with multiple instantiations of the X server, each with its own mouse, keyboard, and heads. Multi-seat is not supported on the HP-UX 9.0x release.

- **Single Logical Screen:** A configuration in which a single X server with a single mouse and keyboard drives multiple homogeneous graphics devices (heads) concurrently while allowing the heads to emulate a large single screen. This differs from a multi-screen environment by allowing windows to be moved and displayed across heads. See the section in this document on Single Logical Screen.

Note that different monitor resolutions are not supported with multi-display configurations.

3

Multi-Screen Support

This section refers to multi-screen configurations only. Running one X server on more than one graphics display is called a “multi-screen” operation. The keyboard and pointer are shared among the screens. Multiple screens are enabled via the `/usr/lib/X11/X*screens` file. The `X*screens` file is used to configure the operation of the X server. The screens are defined in the `X*screens` file by specifying the appropriate special device files. See the section in this document on special device files and `/usr/lib/X11/X0screens` for more information.

A separate screen entry for each graphics display is entered in the `X*screens` file. The order of entries determines each screen number starting at 0. The devices can be arranged in any order.

For example, in the following multi-screen system, the screen numbers are assigned as indicated:

```
/dev/crt1    # first entry is screen 0 (as in local:0.0)
/dev/crt0    # second entry is screen 1 (as in local:0.1)
/dev/crt2    # third entry is screen 2 (as in local:0.2)
```

The X server supports up to four screens at a time. Specifying more than four screens will cause a server error message.

The following multi-screen configurations are supported (unless otherwise stated, different resolutions are not supported with multi-display configurations):

- 712/60 and 712/80:
 - Integrated Color Graphics and one plug-in Color Graphics card.
- 715/33:
 - Internal Color Graphics and one CRX-24
 - Internal Color Graphics and one CRX-24Z
- 715/50, 715/75, 725/50, and 725/75:
 - Internal Color Graphics and one CRX-24
 - Internal Color Graphics and one CRX-24Z
 - Internal Color Graphics and one CRX-48Z
- 715/64, 715/80, and 715/100:
 - Integrated Color Graphics and one plug-in Color Graphics card.
 - Integrated Color Graphics and one Dual Color Graphics card
 - Integrated Color Graphics and one HCRX-24
 - Integrated Color Graphics and one HCRX-24Z
 - Integrated Color Graphics and one HP VISUALIZE-24
- 725/100:
 - Integrated Color Graphics and up to two plug-in Color Graphics cards
 - Integrated Color Graphics and one Dual Color Graphics card
 - Integrated Color Graphics and one HCRX-24
 - Integrated Color Graphics and one HCRX-24Z
 - Integrated Color Graphics and one HP VISUALIZE-24
- 720, 730, 735, and 735/125:
 - One Dual CRX
- 748i/64 and 748i/100:
 - Integrated Color Graphics and one plug-in Color Graphics card (3x5)
 - Two plug-in Color Graphics cards (3x5)
- 747i/50 and 747i/100:
 - One Dual CRX
 - Internal Color Graphics and one Dual CRX
- 750, 755 and 755/125:
 - One Dual CRX
 - Two Dual CRXs
 - Two CRX-24s
- J200 and J210:

- Up to four screens with any combination of plug-in Color Graphics cards and Dual Color Graphics cards
 - One or two Dual Color Graphics cards
 - Up to three plug-in Color Graphics cards
 - One Dual Color Graphics card and up to two plug-in Color Graphics cards
- Two HCRX-24 cards
- One plug-in Color Graphics card and one HCRX-24Z
- One plug-in Color Graphics card and one HP VISUALIZE-24
- One plug-in Color Graphics card and one CRX-48Z



3

3-20 X Windows, HP-UX 9.0x

Integrated Color Graphics Device-Dependent Information

This sections includes information on Integrated Color Graphics, Color Graphics, and Dual Color Graphics cards.

Supported Visuals

For color displays:

- Class `PseudoColor` Depth 8:
supports DBE and MBX software double-buffering
- Class `TrueColor` Depth 8:
supports DBE and MBX software double-buffering

For grayscale displays, only one visual is supported:

- Class `GrayScale` Depth 8:
supports DBE and MBX software double-buffering

Supported Environment Variables

The following environment variables are supported:

- `SWAP_BUFFERS_ON_VBLANK`
- `HP_DISABLE_COLOR_RECOVERY`
- `HP_3_BIT_CENTERCOLOR`
- `HPGCRX_IMAGETEXT_VIA_BITMAP`

Colormaps and Colormap Management

Color Graphics devices have two hardware colormaps (color lookup tables), each with 256 entries. The X server controls the allocation and contents of these hardware colormaps.

The Default Colormap Management Scheme

Many applications use the default X11 colormap. A “technicolor” effect in the windows using the default colormap occurs when a non-default colormap is downloaded into the hardware colormap that had previously contained the default colormap.

Because so many applications use the default X11 colormap—including the window manager—and because Color Graphics devices have two hardware colormaps, the default behavior on this device is to dedicate one hardware colormap to always hold the default X11 colormap. The second hardware colormap is available to applications that use colormaps other than the default.

The default behavior can cause technicolor if two or more applications are using different, non-default colormaps. For example, Application A uses the default X11 colormap, Application B uses a different colormap, and Application C uses a third colormap. If applications A, B, and C are all executed simultaneously on a Model 712, application A would look correct. Either application B or C would have a technicolor effect—the application whose colormap was last downloaded into the hardware colormap would look correct.

Accessing HP Color Recovery Technology via Xlib

Color Recovery is a technique to generate a better picture by attempting to eliminate the graininess caused by dithering. Access to the Color Recovery capability is transparent when using a 3D graphics API such as Starbase, HP-PHIGS or PEX. If you are producing graphics using Xlib calls, your application must perform some of the necessary processing. At server startup (if Color Recovery is not disabled in the X*screens file), the following properties are defined and placed on the root window:

- `_HP_RGB_SMOOTH_TRUE_MAP`
- `_HP_RGB_SMOOTH_PSEUDO_MAP`
- `_HP_RGB_SMOOTH_MAP_LIST`

These properties are of type `RGB_COLOR_MAP` and carry pointers to structures of type `XStandardColormap`. They may be interrogated with calls to `XGetRGBColormaps`. The colormaps in the `_HP_RGB_SMOOTH_TRUE_MAP` and `_HP_RGB_SMOOTH_PSEUDO_MAP` structures identify colormaps which are created at server startup and are for use with the `TrueColor` and `PseudoColor` visuals, respectively. They are both initialized to contain the 3:3:2 ramp of 8-bit `TrueColor`. Neither of these colormaps can be modified, as they are *read-only*. The property `_HP_RGB_SMOOTH_MAP_LIST` is a list of colormaps that are associated with window visual IDs that support Color Recovery. When the `XGetRGBColormaps` routine searches through this list for a colormap with a visual ID that matches your window's visual ID and it finds one, your application knows that

your visual supports Color Recovery, and uses that colormap for any Color Recovery window in your window's visual.

Note that the algorithm used for the Color Graphics device is slightly different from that used for the HCRX family of devices. If you do not wish for your application to have to do device-specific checks, HP recommends that you use the HCRX encoding algorithm for Color Recovery regardless of the device on which your application is executing. The results on the Color Graphics device will not be optimal, but will generally still be much better than a standard dither. If you are willing to do device-specific checks, the existence of either the `_HP_RGB_SMOOTH_TRUE_MAP` or `_HP_RGB_SMOOTH_PSEUDO_MAP` property will indicate the device is Color Graphics.

Color Recovery uses all 256 entries of one of the available colormaps. The color visual used by Color Recovery emulates the 24-bit **TrueColor** visual, thus, the colors red, green, and blue are typically declared as integers in the range from 0 to 255. Note that each window that uses Color Recovery will have the same colormap contents.

For Color Recovery to produce the best results, the emulated 24-bit TrueColor data is dithered as explained below.

A pixel to be dithered is sent to the routine provided in this example. Note that the values of the variables `RedValue`, `GreenValue`, and `BlueValue` are generated by an application. In this example, the color values are assumed to be in the range 0..255.

The given routine receives the color values and the X and Y window address (`Xp` and `Yp`) of the pixel. The X and Y address is used to access the dither tables. The values from the dither tables are added to the color values. After the dither addition, the resultant color values are quantized to three bits of red and green and two bits of blue. The quantized results are packed into an 8-bit unsigned char and then stored in the frame buffer. In the process of sending the contents of the frame buffer to the CRT, a special section in the hardware then converts the frame buffer's 8-bit data into a 24-bit TrueColor data for display.

Here is a routine that can be used to dither the 24-bit TrueColor data.

```
unsigned char dither_pixel_for_CR(RedValue, GreenValue, BlueValue, Xp, Yp)
int RedValue, GreenValue, BlueValue, Xp, Yp;
{
    static short dither_red[2][16] = {
        {-16, 4, -1, 11,-14, 6, -3, 9,-15, 5, -2, 10,-13, 7, -4, 8},
        { 15, -5, 0,-12, 13, -7, 2,-10, 14, -6, 1,-11, 12, -8, 3, -9}};
    static short dither_green[2][16] = {
        { 11,-15, 7, -3, 8,-14, 4, -2, 10,-16, 6, -4, 9,-13, 5, -1},
        {-12, 14, -8, 2, -9, 13, -5, 1,-11, 15, -7, 3,-10, 12, -6, 0}};
    static short dither_blue[2][16] = {
        {-3, 9,-13, 7, -1, 11,-15, 5, -4, 8,-14, 6, -2, 10,-16, 4},
        { 2,-10, 12, -8, 0,-12, 14, -6, 3, -9, 13, -7, 1,-11, 15, -5}};
    int red, green, blue;
    int x_dither_table, y_dither_table;
    unsigned char pixel;

    /* Determine the dither table entries to use based on the pixel address */
    x_dither_table = Xp % 16; /* X Pixel Address MOD 16 */
    y_dither_table = Yp % 2; /* Y Pixel Address MOD 2 */

    /* Start with the initial values as supplied by the calling routine */
    red = RedValue;
    green = GreenValue;
    blue = BlueValue;
    /* Generate the red dither value */
    red += dither_red[y_dither_table][x_dither_table];
    /* Check for overflow or underflow on red value */
    if (red > 0xff) red = 0xff;
    if (red < 0x00) red = 0x00;
    /* Generate the green dither value */
    green += dither_green[y_dither_table][x_dither_table];
    /* Check for overflow or underflow on green value */
    if (green > 0xff) green = 0xff;
    if (green < 0x00) green = 0x00;
    /* Generate the blue dither value */
    blue += (dither_blue[y_dither_table][x_dither_table]<<1);
    /* Check for overflow or underflow on blue value */
    if (blue > 0xff) blue = 0xff;
    if (blue < 0x00) blue = 0x00;
    /* Generate the pixel value by "or"ing the values together */
    pixel = ((red & 0xE0) | ((green & 0xE0) >> 3) | ((blue & 0xC0) >> 6));
    return(pixel);
}
```

3

Internal Color Graphics, Internal GrayScale Graphics, CRX, GRX, and Dual-CRX Device-Dependent Information

Supported Visuals

Only one visual is supported.

For color displays:

- Class `PseudoColor` Depth 8:
 - supports DBE and MBX hardware double-buffering (CRX, Dual CRX);
 - supports DBE and MBX software double-buffering (Internal Color Graphics).

For grayscale displays:

- Class `GrayScale` Depth 8:
 - supports DBE and MBX hardware double-buffering (GRX);
 - supports DBE and MBX software double-buffering (Internal GrayScale Graphics).

Supported Environment Variables

The following environment variables are supported:

- `SWAP_BUFFERS_ON_VBLANK`
- `HP_3_BIT_CENTERCOLOR` (Internal Color Graphics only)
- `HPGCRX_IMAGETEXT_VIA_BITMAP`

CRX-24[Z] Device-Dependent Information

Supported Visuals

The following visuals are supported:

- Class `PseudoColor` Depth 8 Layer Image:
supports DBE and MBX hardware double-buffering
- Class `PseudoColor` Depth 8 Layer Overlay:
supports DBE and MBX software double-buffering
- Class `DirectColor` Depth 12 Layer Image:
supports DBE and MBX hardware double-buffering
- Class `TrueColor` Depth 12 Layer Image:
supports DBE and MBX hardware double-buffering
- Class `DirectColor` Depth 24 Layer Image:
does not support DBE and MBX double-buffering
- Class `TrueColor` Depth 24 Layer Image:
does not support DBE and MBX double-buffering

Supported Environment Variables

The following environment variables are supported:

- `SWAP_BUFFERS_ON_VBLANK`
- `HP_COUNT_TRANSPARENT_IN_OVERLAY_VISUAL`
- `HPGCRX_IMAGETEXT_VIA_BITMAP`

CRX-24[Z] Transparent Overlay Visuals

The default number of colormap entries in the overlay visual for the CRX-24[Z] is 255. Entry 255 is excluded because its value is hard-coded to transparent (that is, show the image planes).

This may have the following two consequences for X11 applications running in the overlay planes (the default visual):

- Clients attempting to allocate 256 entries do not have their request granted.
- Clients requesting (via `XAllocNamedColor`) the `rgb.txt` value of `Transparent` are not returned entry 255.

3-26 X Windows, HP-UX 9.0x

This default behavior can be changed by setting the environment variable `HP_COUNT_TRANSPARENT_IN_OVERLAY_VISUAL` to any value. When this option is enabled, the X server does the following:

- Specifies that the overlay visual has 256 entries.
- Creates the default colormap with entry 255 pre-allocated to `Transparent`. A client calling `XAllocNamedColor` for entry `Transparent` in the default colormap will be returned entry 255.
- For all other colormaps, returns all 256 entries as allocable, but issues a warning message:

```
Warning: XCreateColormap is creating 256 entry cmaps in overlay visual.  
        Though allocable, entry 255 is hard-coded to transparency.
```

This warning is issued once per server execution.

CRX-48Z Device-Dependent Information

Supported Visuals

The following visuals are supported:

- Class PseudoColor Depth 8 Layer Image:
supports DBE and MBX hardware double-buffering
- Class PseudoColor Depth 8 Layer Overlay:
supports DBE and MBX software double-buffering
- Class DirectColor Depth 24 Layer Image:
supports DBE and MBX hardware double-buffering
- Class TrueColor Depth 24 Layer Image:
supports DBE and MBX hardware double-buffering

Supported Environment Variables

The following environment variables are supported:

- SWAP_BUFFERS_ON_VBLANK
- HP_COUNT_TRANSPARENT_IN_OVERLAY_VISUAL
- HPGCRX_IMAGETEXT_VIA_BITMAP

CRX-48Z Transparent Overlay Visuals

The default number of colormap entries in the overlay visual for the CRX-48Z is 255. Entry 255 is excluded because its value is hard-coded to transparent (that is, show the image planes).

This may have the following two consequences for X11 applications running in the overlay planes (the default visual):

- Clients attempting to allocate 256 entries do not have their request granted.
- Clients requesting (via `XAllocNamedColor`) the `rgb.txt` value of `Transparent` are not returned entry 255.

This default behavior can be changed by setting the environment variable `HP_COUNT_TRANSPARENT_IN_OVERLAY_VISUAL` to any value. When this option is enabled, the X server does the following:

- Specifies that the overlay visual has 256 entries.
- Creates the default colormap with entry 255 pre-allocated to `Transparent`. A client calling `XAllocNamedColor` for entry `Transparent` in the default colormap will be returned entry 255.
- For all other colormaps, returns all 256 entries as allocable, but issues a warning message:

```
Warning: XCreateColormap is creating 256 entry cmaps in overlay visual.  
        Though allocable, entry 255 is hard-coded to transparency.
```

This warning is issued once per server execution.

HCRX Device-Dependent Information

This section includes information on the HCRX-8[Z] and HCRX-24[Z] devices, and the HP VISUALIZE-8, HP VISUALIZE-24, and HP VISUALIZE-48 devices.

The HCRX-8[Z] is a one board device (two, with the optional accelerator that has eight overlay planes, two banks of 8 image planes, and 4 hardware colormaps. This device provides a superset of functionality in the CRX.

The HCRX-24[Z] is a one board device (two, with the optional accelerator) that has eight overlay planes, two banks of 12 image planes, and 4 hardware colormaps. This device provides a superset of functionality in the CRX-24[Z].

The HP VISUALIZE-8 is a two board accelerated device that has eight overlay planes, two banks of 8 image planes, and 4 hardware colormaps. This device provides a superset of functionality in the CRX.

The HP VISUALIZE-24 is a two board accelerated device that has eight overlay planes, two banks of 12 image planes, and 4 hardware colormaps. This device provides a superset of functionality in the CRX-24[Z].

The HP VISUALIZE-48 is a two-board accelerated device (three, with the optional texture- mapping hardware) that has eight overlay planes, two banks of 24 image planes, and six hardware colormaps. This device provides a superset of functionality in the CRX-48Z. The hardware support for accelerating 2D Xlib primitives is similar to that in the other HCRX devices. The hardware for accelerating 3D geometry, lighting, and shading, is new.

Supported Visuals

The following visuals are supported on the HCRX-8[Z] and HP VISUALIZE-8:

- Class `PseudoColor` Depth 8 Layer Image:
supports DBE and MBX hardware double-buffering
- Class `PseudoColor` Depth 8 Layer Overlay:
(see Note) supports DBE and MBX software double-buffering
- Class `PseudoColor` Depth 8 Layer Overlay Transparent:
(see Note) supports DBE and MBX software double-buffering
- Class `TrueColor` Depth 8 Layer Image:
supports DBE and MBX hardware double-buffering

Note



The two overlay visuals are mutually exclusive, based on the presence of the `HP_ENABLE_OVERLAY_TRANSPARENCY` environment variable (i.e., if the `HP_ENABLE_OVERLAY_TRANSPARENCY` environment variable is set, then the visual that supports transparency *is* available, otherwise the visual which does *not* support transparency is available).

The following visuals are supported on the HCRX-24[Z] and HP VISUALIZE-24:

- Class `PseudoColor` Depth 8 Layer Image:
supports DBE and MBX hardware double-buffering
- Class `PseudoColor` Depth 8 Layer Overlay:
supports DBE and MBX software double-buffering
- Class `PseudoColor` Depth 8 Layer Overlay Transparent:
supports DBE and MBX software double-buffering
- Class `TrueColor` Depth 8 Layer Image:
supports DBE and MBX hardware double-buffering
- Class `DirectColor` Depth 12 Layer Image:
supports DBE and MBX hardware double-buffering
- Class `TrueColor` Depth 12 Layer Image:
supports DBE and MBX hardware double-buffering
- Class `DirectColor` Depth 24 Layer Image:
does not support DBE and MBX double-buffering
- Class `TrueColor` Depth 24 Layer Image:
does not support DBE and MBX double-buffering



The following visuals are supported on the HP VISUALIZE-48:

- Class `PseudoColor` Depth 8 Layer Image:
supports DBE and MBX hardware double-buffering
- Class `PseudoColor` Depth 8 Layer Overlay:
supports DBE and MBX software double-buffering
- Class `PseudoColor` Depth 8 Layer Overlay Transparent:
supports DBE and MBX software double-buffering
- Class `TrueColor` Depth 8 Layer Image:
supports DBE and MBX hardware double-buffering
- Class `DirectColor` Depth 24 Layer Image:
supports DBE and MBX hardware double-buffering
- Class `TrueColor` Depth 24 Layer Image:
supports DBE and MBX hardware double-buffering

Supported Environment Variables

The following environment variables are supported:

- `SWAP_BUFFERS_ON_VBLANK`
- `HP_DISABLE_COLOR_RECOVERY`
- `HP_COUNT_TRANSPARENT_IN_OVERLAY_VISUAL`
- `HP_ENABLE_OVERLAY_TRANSPARENCY` (HCRX-8[Z] and HP VISUALIZE-8 only)
- `HPGCRX_IMAGETEXT_VIA_BITMAP`

HCRX Configuration Hints

HCRX-8[Z] and HP VISUALIZE-8 Visuals and Double-Buffer Support

The 8-plane HCRX-8[Z] and HP VISUALIZE-8 are the first members of the Series 700 graphics family whose overlay planes and image planes are both depth 8.

- There are two depth-8 `PseudoColor` visuals (one in the overlay planes, the other in the image planes). There is also a depth-8 `TrueColor` visual in the image planes.
- The default visual (where the root window and default colormap reside) is in the overlay planes.
- Fast 8/8 double-buffering (two hardware buffers) is supported in the depth-8 image planes, but not in the overlays. The overlay planes support the slower virtual-memory-based double-buffering.

3-32 X Windows, HP-UX 9.0x

Implications and Suggestions for HCRX-8[Z] and HP VISUALIZE-8

The default colormap cannot be used with a window in a non-default visual, even one of the same depth as the default visual.

Before trying to use the default colormap in a depth-8 window, verify that the window is in the default visual. If the window is not in the default visual, create a colormap in that visual. This process is the same as the one used to create windows in depth-12 or depth-24 visuals.

Unlike the CRX, the HCRX-8[Z]'s default visual and the HP VISUALIZE-8's default visual do not have fast hardware double-buffering (but the image planes do).

To obtain hardware double-buffering, find a visual in the image planes. The best method is to find all the depth-8 `PseudoColor` visuals returned by `XGetVisualInfo` and then eliminate the visuals that are reported in the `SERVER_OVERLAY_VISUALS` property (discussed below).

HCRX Overlay Visuals and Overlay Transparency

As on the CRX-24[Z] and CRX-48Z, the `SERVER_OVERLAY_VISUALS` property on the root window is used to describe the visuals that are in the overlay planes.

Overlay Transparency on the HCRX-8[Z] and HP VISUALIZE-8

The 8-plane HCRX-8[Z] and the 8-plane HP VISUALIZE-8 both have one visual in the overlay planes (depth-8 `PseudoColor`). By default, these overlay visuals have no transparent index available to applications for rendering transparency. This means the overlay windows with “floating text” are not supported in the typical X server operation on the HCRX-8[Z] or HP VISUALIZE-8.

For applications that require transparent overlay windows on the HCRX-8[Z] or HP VISUALIZE-8, an optional X server mode is available to allow for overlay transparency, but it is restrictive. In this optional mode, overlay colormaps provide a single entry that can be used to render transparency. Only one hardware colormap is available in the overlays (instead of two) and only one hardware colormap is available in the image planes (instead of two).

To activate this optional X server mode to enable transparency, set the environment variable `HP_ENABLE_OVERLAY_TRANSPARENCY` to any value. You will need to restart the X server for the option to take effect.

With this mode enabled, colormaps created in the default visual have 255 entries; entry 256 is reserved for transparency. As on the CRX-24[Z] and CRX-48Z, the environment variable `HP_ENABLE_OVERLAY_TRANSPARENCY` can be used to include the transparent index in the colormap size (256 entries instead of 255).

Programmer's Note If transparency is not enabled, there are only 252 colors available. Entries 252-255 are not writable, and should not be used; there are only 252 colormap entries available, even though the server states that there are 256.

3

Overlay Transparency on the HCRX-24[Z], HP VISUALIZE-24, and -48

The HCRX-24[Z], HP VISUALIZE-24, and HP VISUALIZE-48 have two visuals in the overlay planes, both depth-8 PseudoColor.

The default overlay visual has 256 entries per colormap and no transparency.

The second overlay visual has 255 entries per colormap and supports transparency in the same way as the CRX-24[Z] and CRX-48Z. As on the CRX-24[Z] and CRX-48Z, the environment variable `HP_ENABLE_OVERLAY_TRANSPARENCY` can be used to include the transparent index in the colormap size (256 entries instead of 255).

To allow applications to determine which visuals are in the overlay planes, both overlay visuals are listed in the `SERVER_OVERLAY_VISUALS` property attached to the root window. The default overlay visual has a transparent type of "0" (`None`) while the transparent overlay visual has a transparent type of "1" (`TransparentPixel`).

If you need an overlay colormap that supports transparency, create the colormap using the visual that has transparency in its `SERVER_OVERLAY_VISUALS` property. To look at the contents of this property, you would use code similar to the following:

```

{
typedef struct {
    VisualID    overlayVisualID;
    Card32      transparentType; /* None, TransparentPixel, TransparentMask */
    Card32      value;          /* Either pixel value or pixel mask */
    Card32      layer;
} OverlayVisualPropertyRec;

OverlayVisualPropertyRec *pOverlayVisuals, *p0Vis;
XVisualInfo              getVis;
XVisualInfo              *pVisuals;
Atom                    overlayVisualsAtom, actualType;
...
/* Get the visuals for this screen and allocate. */
getVis.screen = screen;
pVisuals = XGetVisualInfo(display, VisualScreenMask, &getVis, &nVisuals);
pOverlayVisuals = (OverlayVisualPropertyRec *)
    malloc ( (size_t)nVisuals * sizeof(OverlayVisualPropertyRec) );

/* Get the overlay visual information for this screen. Obtain
 * this information from the SERVER_OVERLAY_VISUALS property. */
overlayVisualsAtom = XInternAtom(display, "SERVER_OVERLAY_VISUALS", True);
if (overlayVisualsAtom != None)
{
    /* Since the Atom exists, request the property's contents. */
    bytesAfter = 0;
    numLongs = ( nVisuals * sizeof(OverlayVisualPropertyRec) + 3 ) / 4;
    XGetWindowProperty(display, RootWindow(display, screen),
        overlayVisualsAtom, 0, numLongs, False,
        AnyPropertyType, &actualType, &actualFormat,
        &numLongs, &bytesAfter, &pOverlayVisuals);
    if ( bytesAfter != 0 ) { /* Serious Failure Here */ } ;
}

```

```

/* Loop through the pOverlayVisuals array. */
...
n0Visuals = numLongs/sizeof(OverlayVisualPropertyRec);
p0Vis = pOverlayVisuals;
while (--n0Visuals >= 0)
{
    if ( p0Vis->transparentType == TransparentPixel )
        /* Found a transparent overlay visual, set ident. aside. */;
    p0Vis++;
}
XFree(pOverlayVisuals);
/* There might be some additional checking of the found
transparent overlay visuals wanted; e.g., for depth. */
}
XFree(pVisuals);
}

```

3

This program fragment is not complete; its main purpose is to give the idea of how to find an overlay visual having transparency.

HCRX Colormaps

The following information discusses the number of supported colormaps for the HCRX configurations.

HCRX-8[Z] and HP VISUALIZE-8: Eight Overlay Planes and Two Depth-8 Banks of Image Planes. When environment variable `HP_ENABLE_OVERLAY_TRANSPARENCY` is not set, the overlay planes contain the default colormap permanently installed in the hardware, plus one other hardware colormap available to applications. The image planes contain two hardware colormaps each usable by applications.

When the environment variable `HP_ENABLE_OVERLAY_TRANSPARENCY` is set, both the overlay planes and the image planes have access to one hardware colormap. The default colormap is not permanently installed in the hardware.

HCRX-24[Z] and HP VISUALIZE-24: 8 Overlay Planes and 24 Image Planes. The overlay planes contain the default colormap permanently installed in the hardware, plus one other hardware colormap available to applications. The image planes contain two hardware colormaps, each usable by applications.

Although two hardware colormaps are available to applications in the image planes, a hardware restriction allows only one depth-12 or depth-24 colormap to be installed at any given time. Therefore, if two applications are run simultaneously and use different depth-12 or depth-24 colormaps, the application that has the colormap focus looks correct and the other is technicolored.

3-36 X Windows, HP-UX 9.0x

HP VISUALIZE-48: Eight Overlay Planes and 48 Image Planes. The overlay planes contain the default colormap permanently installed in the hardware, plus one other hardware colormap available to applications. The image planes contain four hardware colormaps, each usable by applications.

The four hardware colormaps in the image planes can be treated as depth-8 or depth-24 colormaps. There are no restrictions on the types of colormaps that can be installed in the hardware at any given time. All four colormaps can be used with any visual class.

Accessing HP Color Recovery Technology via Xlib

Color Recovery is a technique to generate a better picture by attempting to eliminate the graininess caused by dithering. Access to the Color Recovery capability is transparent when using a 3D graphics API such as Starbase, HP-PHIGS or PEX. If you are producing graphics using Xlib calls, your application must perform some of the necessary processing. At server startup (if Color Recovery is not disabled in the `X*screens` file), the `_HP_RGB_SMOOTH_MAP_LIST` property is defined and placed on the root window. The above property is of type `RGB_COLOR_MAP` and carries pointers to structures of type `XStandardColormap`. It may be interrogated with calls to `XGetRGBColormaps`. The property `_HP_RGB_SMOOTH_MAP_LIST` is a list of colormaps that are associated with window visual IDs that support Color Recovery. When the `XGetRGBColormaps` routine searches through this list for a colormap with a visual ID that matches your window's visual ID and it finds one, your application knows that your visual supports Color Recovery, and uses that colormap for any Color Recovery window in your window's visual.

Color Recovery uses all 256 entries of one of the available colormaps. The color visual used by Color Recovery emulates the 24-bit `TrueColor` visual, thus, the colors red, green, and blue are typically declared as integers in the range from 0 to 255. Note that each window that uses Color Recovery will have the same colormap contents.

For Color Recovery to produce the best results, the emulated 24-bit `TrueColor` data is dithered as explained below.

A pixel to be dithered is sent to the routine provided in this example. Note that the values of the variables `RedValue`, `GreenValue`, and `BlueValue` are generated

by an application. In this example, the color values are assumed to be in the range 0..255.

The given routine receives the color values and the X and Y window address (**Xp** and **Yp**) of the pixel. The X and Y address is used to access the dither tables. The values from the dither tables are added to the color values. After the dither addition, the resultant color values are quantized to three bits of red and green and two bits of blue. The quantized results are packed into an 8-bit unsigned char and then stored in the frame buffer. In the process of sending the contents of the frame buffer to the CRT, a special section in the hardware then converts the frame buffer's 8-bit data into a 24-bit **TrueColor** data for display.



3

Here is a routine that can be used to dither the 24-bit TrueColor data.

```

unsigned char dither_pixel_for_CR(RedValue, GreenValue, BlueValue, Xp, Yp)
int          RedValue, GreenValue, BlueValue, Xp, Yp;
{
    static short    dither_red[2][16] = {
        {-16,  4, -1, 11,-14,  6, -3,  9,-15,  5, -2, 10,-13,  7, -4,  8},
        { 15, -5,  0,-12, 13, -7,  2,-10, 14, -6,  1,-11, 12, -8,  3, -9}};
    static short    dither_green[2][16] = {
        { 11,-15,  7, -3,  8,-14,  4, -2, 10,-16,  6, -4,  9,-13,  5, -1},
        {-12, 14, -8,  2, -9, 13, -5,  1,-11, 15, -7,  3,-10, 12, -6,  0}};
    static short    dither_blue[2][16] = {
        {-3,  9,-13,  7, -1, 11,-15,  5, -4,  8,-14,  6, -2, 10,-16,  4},
        { 2,-10, 12, -8,  0,-12, 14, -6,  3, -9, 13, -7,  1,-11, 15, -5}};
    int             red, green, blue;
    int             x_dither_table, y_dither_table;
    unsigned char   pixel;

    /* Determine the dither table entries to use based on the pixel address */
    x_dither_table = Xp % 16; /* X Pixel Address MOD 16 */
    y_dither_table = Yp % 2; /* Y Pixel Address MOD 2 */

    /* Start with the initial values as supplied by the calling routine */
    red  = RedValue;
    green = GreenValue;
    blue = BlueValue;
    /* Generate the red dither value */
    if (red >= 48) /* 48 is a constant required by this routine */
        red=red-16;
    else
        red=red/2+8;
    red += dither_red[y_dither_table][x_dither_table];
    /* Check for overflow or underflow on red value */
    if (red > 0xff) red = 0xff;
    if (red < 0x00) red = 0x00;

    /* Generate the green dither value */
    if (green >= 48) /* 48 is a constant required by this routine */
        green=green-16;
    else
        green=green/2+8;
    green += dither_green[y_dither_table][x_dither_table];
    /* Check for overflow or underflow on green value */
    if (green > 0xff) green = 0xff;
    if (green < 0x00) green = 0x00;
}

```

```
        /* Generate the blue dither value */
if (blue >= 112) /* 112 is a constant required by this routine */
    blue=blue-32;
else
    blue=blue/2+24;
blue += (dither_blue[y_dither_table][x_dither_table]<<1);
/* Check for overflow or underflow on blue value */
if (blue > 0xff) blue = 0xff;
if (blue < 0x00) blue = 0x00;
pixel = ((red & 0xE0) | ((green & 0xE0) >> 3) | ((blue & 0xC0) >> 6));
return(pixel);
}
```

 3

Freedom Series Graphics (S3150, S3250 and S3400) Device-Dependent Information

This sections describes support for the Freedom Series on Hewlett-Packard workstations.

Supported Visuals

The following visuals are supported:

- Class `PseudoColor` Depth 8 Layer Image:
supports DBE and MBX hardware double-buffering
- Class `DirectColor` Depth 24 Layer Image:
supports DBE and MBX hardware double-buffering
- Class `TrueColor` Depth 24 Layer Image:
supports DBE and MBX hardware double-buffering
- Class `PseudoColor` Depth 8 Layer Overlay:
supports DBE and MBX software double-buffering

3

Supported Environment Variables

No device-specific environment variables are supported.

GLX

GLX is the OpenGL Extension to the X Window System. It is supported only on the Freedom Series Graphics Devices when attached to HP systems.

VRX Device-Dependent Information

This section includes information on the PersonalVRX (PVRX) and TurboVRX (TVRX) graphics devices.

Supported Visuals

The following visuals are supported:

- Depth 3 (overlay and combined mode)
- Depth 4 (overlay and combined mode)
- Depth 8 (image and combined mode)
- Depth 12 (image and combined mode, TVRX only)
- Depth 16 (Creates a double-buffer version of the Depth 8 visual)
- Depth 24 (image and combined mode, TVRX only)

None of these visuals support DBE and MBX double-buffering.

In image mode, the default visual is the Depth 8 `PseudoColor` visual. In overlay mode it is the depth 3 or depth 4 `PseudoColor` visual as specified by the device file. In combined mode the first device file specifies the default visual. Examples are shown in the section below.

VRX Device Files

Different device files exist for the image planes and overlay planes on VRX devices. The following table shows examples of device files for VRX devices:

Table 3-4. mknod Information for HP-UX 9.x

Device Filename	9.0x Major Number	10.0 Major Number	Minor Number	Description
<code>/dev/crt</code>	12	174	0x000000	Image mode
<code>/dev/ocrt</code>	12	174	0x000001	Overlay mode (3 planes)
<code>/dev/o4crt</code>	12	174	0x000003	Overlay mode (4 planes)

The X server supports three different modes of operation on VRX devices: image, overlay and combined.

3-42 X Windows, HP-UX 9.0x

In image mode, the X server runs only in the image planes. This is the default on VRX devices. To operate in image mode, the image device file should be specified as the primary screen device. For example:

```
/dev/crt                # Image mode
```

In overlay mode, the X server runs only in the overlay planes. Since only 3 or 4 planes are available in the overlay planes on VRX devices, the number of colors is very limited. To operate in overlay mode, the overlay device file should be specified as the primary screen device. For example:

```
/dev/ocrt                # Overlay mode using 3 overlay planes  
    or  
/dev/o4crt                # Overlay mode using 4 overlay planes
```

In combined mode, the X server runs in both image and overlay planes. To configure the X server to operate in combined mode, a primary and a secondary device must be specified. The `VRXSecondaryDevice` is used for this purpose. For example:

```
/dev/ocrt /dev/crt        # default visual lives in overlay planes  
    or  
/dev/crt /dev/ocrt        # default visual lives in image planes
```



X Windows: HP-UX 10.x

This chapter documents information specific to the HP X server. It describes features unique to HP's X server, provides information on how to configure the X server and includes a list of supported X configurations. For each supported graphics device, device-dependent configuration information is provided.

Information specific to a new release of the X server, beyond the scope of the general information in this document, can be found in the HP-UX Release Notes located in `/usr/share/doc`.

If you prefer to read this information on paper, see the *Graphics Administration Guide*. It includes the same information as is contained here in this on-line document.

The X*screens File

The `X*screens` files are used to configure the operation of the X server. These files may be configured manually or through SAM. For manual changes please refer to the sample file in `/etc/X11/X0screens` for more information on how to use the `X*screens` files (one is included here for reference).

Description of the X*screens Configuration File

This file belongs in `/etc/X11/X*screens`, where “*” is the display number of the server. For example, the “X0screens” file is used when the `$DISPLAY` environment variable is set to `{hostname}:0.{screen}` and the server is invoked using the “:0” option.

The X*screens file is used to specify:

- Device-independent server options, and
- For each screen:
 - What device file to use (required),
 - The default visual,
 - Monitor size, and
 - Device-dependent screen options.

Note that all of the items above, except for device-independent server options, are specified on a per-screen basis.

The X server supports up to four screens at a time. Specifying more than four screens will cause a server error message.

Syntax Guidelines

- Blank lines and comments (text following “#”) are ignored.
- Entries can occupy more than a single line.
- All symbols in the file are recognized case-*insensitive*.

4

The X*screens File Format

Items must appear in the X*screens file in the order that they are specified below.

```
[ServerOptions
  <server_option>
  ⋮
  <server_option>]
{Screen <device_name>} ||
{SingleLogicalScreen <nRows> <nCols> <device_name1> ... <device_nameN>}
[DefaultVisual
  [Class <visual_class>]
  [Depth <depth>]
  [Layer <layer>]
  [Transparent]]
[MonitorSize <diagonal_length> <units>]
[MinimumMonitorPowerSaveLevel <level>]
[ScreenOptions
  <screen_option>
  ⋮
  <screen_option>]
```

4

Brackets (“[” and “]”) denote optional items. Italicized items in angle brackets (“<” and “>”) denote values to be specified. The double vertical line (“||”) denotes that *one* of the *ored* values (items surrounded by braces, “{” and “}”) must be included.

The block from the “Screen <device_name>” line to the final “<screen_option>” line is referred to as either a “Screen Entry” or as a “Single Logical Screen entry”.

As shown above, the X*screens format is composed of an optional block specifying device-independent server options followed by one or more either Screen or Single Logical Screen entries (maximum of four graphics devices).

The minimum X*screens file is a line with the keyword “Screen” followed by a screen device file. For example:

```
Screen /dev/crt
```

Server Options

For more information about server options, or about additional server options, look in an information file (e.g., `/usr/lib/X11/Xserver/info/screens/hp`).

- **GraphicsSharedMemorySize** *<memory_size>*
Specify the size of the graphics shared memory region. The size must be specified in bytes and must be in hexadecimal.
Default value: 0x580000
Environment Variables Replaced: GRM_SIZE, WMSHMSPC.

- **SMTSizes** *<size_spec>*
The size of the SMT regions (see the Shared Memory Transport section).
Default value: 100000,90000,90000

- **FileDescriptors** *<number>*
The number of file descriptors available to the X server for its use. The number of connections (clients, more or less) is limited by the number of file descriptors.

The minimum value is 25, and a current maximum (as of HP-UX 10.20) of 384, allowing a maximum of slightly under 256 total connections to the server. The default value is 192 (which allows a few under 128 connections). If a value provided is out of range, the server yields a warning and continues using the minimum or maximum, as appropriate. There is, however, a limit of 128 clients that can connect.

The command line option `-lf <number>` also specifies the value.

- **ServerMode** `XPrint|XVideo`
This server option places the X server in either `XPrint` or `XVideo` mode, `XVideo` being the default behavior. `XPrint` is an X extension supporting management of networked printers. Use of the `XPrint` mode disables the normal video output mode of the X server. It is necessary to start two X servers to have both functionalities. To do this, run each invocation with a different display identifier.

- **ConfigXPrintServer** *<path_file_name>*
This server option identifies a configuration file for use by the `XPrint` extension. `ConfigXPrintServer` is only meaningful when used in conjunction with the server option “`ServerMode XPrint`”.

The command line option `-Xpfile <path_file_name>` is an alternate method to specify the same information.

4-4 X Windows: HP-UX 10.x

- **ImmediateLoadDles**

The Xserver delays loading of some X extensions until the first protocol request to the given extension is received. Specifying this server option forces all extensions to be loaded at X server startup. Immediate loading of X extensions is the historical behavior of the HP-UX 10.10 and 10.20 X servers.

Screen Entries

The minimum screen entry is a line with the keyword “**Screen**” followed by a screen device file.

Optional specifications for default visual, monitor size, and device-dependent screen options may follow this minimal screen description line.

- **DefaultVisual**

This optional part of the format specifies the default visual that the screen uses. Valid keywords following the “**DefaultVisual**” keyword are “**Class**”, “**Depth**”, “**Layer**”, and “**Transparent**”.

If no default visual is specified, then the standard default visual class, depth, layer, and transparency for the graphics device is used.

Not all default visual specifications will work on all devices.

If there is an error in a specification, look in an information file for more details (for example, `/usr/lib/X11/Xserver/info/screens/hp`), in case it is newer than the document you’re now reading.

- **Class** *<StaticGray>* | *<GrayScale>* | *<StaticColor>* | *<PseudoColor>* | *<TrueColor>* | *<DirectColor>*

Specify the class of the default visual.

- **Depth** *<depth_value>*

Specify the depth of the default visual (for example 8, 12, or 24).

- **Layer** *<Image>* | *<Overlay>*

Specify the layer of the default visual.

- **Transparent** Specify that a visual with an application-accessible transparent entry in the default colormap be used.

Specifications in the “**DefaultVisual**” section, except for “**Depth**”, are ignored on VRX devices. See the “**ScreenOptions**” section below for VRX-related options.

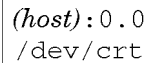
- **MonitorSize** *<diagonal_length>* Inches | MM;
Specify the diagonal size of the monitor. After the “MonitorSize” keyword, you must specify the diagonal length of the monitor and then the units. Use this entry only if you are using a non-standard monitor.
- **MinimumMonitorPowerSaveLevel** *<value>*
Specify the minimum power save level to be used by the monitor during screen blanking. You must specify a level of 0—3. If the option is not used, the default is level 0. On devices that do not support DPMS, this option will be ignored.
- **ScreenOptions**
Screen options are device-dependent options documented in a file in the X server information directory (e.g., `/usr/lib/X11/Xserver/info/screens/hp`).

Sample X*screens Files

Below are several sample X*screens files that illustrate the new format.

- This is the minimum legal X*screens file: the “Screen” keyword followed by the screen device. Since no other information is given, the X server will assume default values for other options and settings.

```
Screen /dev/crt
```

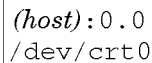


```
(host):0.0  
/dev/crt
```

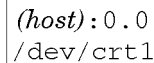
Figure 4-1. Results of Minimal Legal X*screens File

- This is the minimum specification for a two-screen configuration. The maximum number of screens supported on the X server is four. Here, the displays associated with /dev/crt0 and /dev/crt1 are referred to as “$\langle host \rangle:0.0$” and “$\langle host \rangle:0.1$”, respectively.

```
Screen /dev/crt0  
Screen /dev/crt1
```



```
(host):0.0  
/dev/crt0
```



```
(host):0.0  
/dev/crt1
```

Figure 4-2. Two Physical Displays, Two Separate Screens

- This sample `X*screens` file could be used on a system using Internal Color Graphics with a 17-inch monitor. In this example, the `GraphicsSharedMemorySize` is decreased to 1 Mbyte in order to reduce the swap space requirements of the system. Decreasing `GraphicsSharedMemorySize` is appropriate when you do not intend on running any 3D graphics applications.

```
ServerOptions
  GraphicsSharedMemorySize 0x100000
Screen /dev/crt
  MonitorSize 17 inches
```

The display diagram would be the same as that of the “Results of Minimal Legal `X*screens` File” configuration, above.

- This sample `X*screens` file could be used on a system with a CRX24 graphics device. The overlay visual is selected as the default. There are 255 overlay colormap entries available on the CRX24. The 256th entry is hard-wired to transparent. Having less than 256 colormap entries should not cause a problem for most applications, but for those applications that require 256 colormap entries, the `CountTransparentInOverlayVisual` screen option should be used as shown below. Note that any attempts to modify the 256th entry will have no effect on the colormap.

```
Screen /dev/crt
  ScreenOptions
    CountTransparentInOverlayVisual
```

The display diagram would be the same as that of the “Results of Minimal Legal `X*screens` File” configuration, above.

- This sample `X*screens` file could be used on a system with a HCRX-24 graphics device. The default visual on the HCRX-24 is the opaque overlay visual. All 256 colormap entries are opaque and allocable. If an application requires transparency in the default visual, the “`Transparent`” keyword can be used to select the transparent overlay visual as shown below.

```
Screen /dev/crt
  DefaultVisual
    Transparent
```

The display diagram would be the same as that of the “Results of Minimal Legal `X*screens` File” configuration, above.

4-8 X Windows: HP-UX 10.x

- This sample `X*screens` file could be used on a system with a HCRX-8 graphics device. By default on the HCRX-8, the overlay visual does not have a transparent entry available to applications for rendering transparency. If an application requires overlay transparency, an optional X server mode is available, but it is restrictive. In this optional mode, only one hardware colormap is available in the overlays (instead of two) and only one hardware colormap is available in the image planes (instead of two). The optional X server mode can be set via the `EnableOverlayTransparency` screen option as shown below.

```
Screen /dev/crt
  ScreenOptions
    EnableOverlayTransparency
```

The display diagram would be the same as that of the “Results of Minimal Legal `X*screens` File” configuration, above.

- This sample `X*screens` file could be used on a system using either a PVRX or TVRX graphics device. The server will run in combined mode with the default visual residing in the overlay planes. All visual depths which are supported by the graphics device will be available.

```
Screen /dev/ocrt
  ScreenOptions
    VRXSecondaryDevice /dev/crt
```

<pre>(host):0.0 /dev/crt /dev/ocrt</pre>
--

Figure 4-3. PVRX/TVRX Display with Overlays

- This sample `X*screens` file could also be used on a system using PVRX or TVRX graphics. The server will run in combined mode with the default visual in the overlay planes and an 8/8 double-buffered visual in the image planes. In general, specify `VRXDoubleBuffer` if applications will be using DHA (Direct Hardware Access) double-buffer functionality (e.g., Starbase double buffering).

```
Screen /dev/ocrt
  ScreenOptions
    VRXSecondaryDevice /dev/crt
    VRXDepth           16
    VRXDoubleBuffer
```

The display diagram would be the same as that of the “PVRX/TVRX Display with Overlays” configuration, above.

- These sample `X*screens` file entries could be used on a system with two homogeneous graphics devices. Assuming the first device is associated with the device file “`/dev/crt0`” and the second device is associated with the device file “`/dev/crt1`”, both examples specify a horizontal Single Logical Screen configuration.

```
SingleLogicalScreen 1 2
  /dev/crt0 /dev/crt1
  or
SingleLogicalScreen 1 2
  /dev/crt0
  /dev/crt1
```

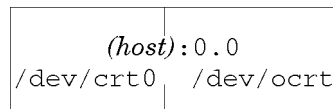


Figure 4-4. Two Physical Displays, Single Logical Screen (1×2)

- These sample `X*screens` entries could be used on a system with four homogeneous graphics devices. Assuming the first device is associated with the device file `“/dev/crt0”`, the second device is associated with the device file `“/dev/crt1”`, etc. The following examples specify valid Single Logical Screen configurations.

- `SingleLogicalScreen 1 4`
`/dev/crt0 /dev/crt1 /dev/crt2 /dev/crt3`

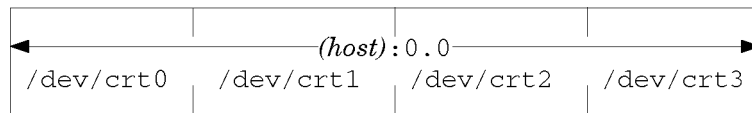


Figure 4-5. Four Physical Displays, Single Logical Screen (1×4)

- SingleLogicalScreen 4 1
 - /dev/crt0
 - /dev/crt1
 - /dev/crt2
 - /dev/crt3



Figure 4-6. Four Physical Displays, Single Logical Screen (4×1)

- `SingleLogicalScreen 2 2`
`/dev/crt0 /dev/crt1`
`/dev/crt2 /dev/crt3`

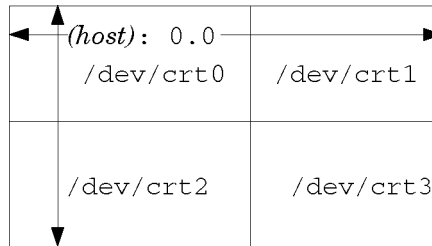


Figure 4-7. Four Physical Displays, Single Logical Screen (2×2)

- It is possible to include a Screen Entry and an SLS Screen Entry in the same `X*screens` File. This creates a situation where there are two X Screens (e.g. `<host>:0.0` and `<host>:1.0`), one of which happens to be a Single Logical Screen. Below is an example of this:

```
Screen /dev/crt0
SingleLogicalScreen 1 2
  /dev/crt1 /dev/crt2
```

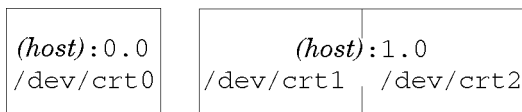


Figure 4-8. Three Physical Displays, Screen plus Single Logical Screen (1×2)

Miscellaneous Topics

Double Buffer Extension (DBE)

DBE is an extension to the X server that provides a double-buffering Application Programming Interface (API). Note that MBX (the **M**ulti-**B**uffering **eX**tension to X) has not been adopted as an industry standard, as DBE has. Thus, it is recommended that applications that use MBX be ported to DBE usage in preparation for future MBX obsolescence (HP-UX 11.0). For more information about DBE and the API, consult the DBE man pages:

- DBE
- XdbeQueryExtension
- XdbeGetVisualInfo
- XdbeFreeVisualInfo
- XdbeAllocateBackBufferName
- XdbeDeallocateBackBufferName
- XdbeSwapBuffers
- XdbeBeginIdiom
- XdbeEndIdiom
- XdbeGetBackBufferAttributes

Performing Buffer Swaps On Vertical Blank

For performance reasons, the default DBE behavior is to *not* synchronize buffer swaps with the monitor's vertical retrace period. In some instances, therefore, image tearing (seeing part of the old image and part of the new image on the display at the same time) could be visible while swapping large DBE windows. For those instances where tearing would occur and is undesirable, an optional X server mode is available to allow for synchronization of buffer swaps with vertical retrace. To activate this optional X server mode, set the following screen option in the `X*screens` File before the X server is started:

```
SwapBuffersOnVBlank
```

Note that `MBX_SWAP_BUFFERS_ON_VBLANK` is obsolete with this release. The `SwapBuffersOnVBlank` Screen Option works for both DBE and MBX.

4-14 X Windows: HP-UX 10.x

Determining Swap Performance

The DBE API does not allow users to determine if double-buffering in a visual is through software or hardware. However, the API does provide a way to determine relative swapping performance on a per-visual basis. The `XdbeScreenVisualInfo()` function returns information about the swapping performance levels for the double-buffering visuals on a display. A visual with a higher performance level is likely to have better double-buffer graphics performance than a visual with a lower performance level. Nothing can be deduced from any of the following: the magnitude of the difference of two performance levels, a performance level in isolation, or comparing performance levels from different servers.

For more information, refer to the DBE man page on `XdbeScreenVisualInfo()`.

Supported Devices

The X server supports DBE on the following devices:

- Internal Color Graphics
- Integrated Color Graphics
- CRX-24[Z]
- CRX-48Z
- HCRX-8[Z]
- HCRX-24[Z]
- HP VISUALIZE-EG
- HP VISUALIZE-8
- HP VISUALIZE-24
- HP VISUALIZE-48
- Freedom Series™ Graphics (S3150, S3250 and S3400)

Display Power Management Signaling (DPMS)

Monitors constitute a large percentage of the power used by a workstation even when not actively in use (i.e., during screen blanking). In order to reduce the power consumption, the Video Electronic Standards Association (VESA) has defined a Display Power Management Signaling (DPMS) standard which can be used to greatly reduce the amount of power being used by a monitor during screen blanking.

The X server features the ability to make use of DPMS on the following graphics devices:

- HCRX: HCRX-8[Z], HCRX-24[Z], HP VISUALIZE-EG, HP VISUALIZE-8, HP VISUALIZE-24, and HP VISUALIZE-48.

The following table is a description of the states that are defined by VESA. The Power Savings column indicates (roughly) the level of power savings achieved in the given state. The Recovery Time is the amount of time that the screen takes to return to a usable state when the screen saver is turned off (by pressing a key or the moving the mouse).

Table 4-1. Power-Saving States Defined by VESA

Level	State	DPMS Compliance Requirements	Power Savings	Recovery Time
0	Screen Saver	Not Applicable	None	Very Short (<1 sec)
1	Stand-by	Optional	Minimal	Short
2	Suspend	Mandatory	Substantial	Longer
3	Off	Mandatory	Maximum	System Dependent

The actual amount of power saved and the recovery time for each of the states is monitor-dependent and may vary widely. The customer can compensate for this by choosing an appropriate level for the monitor that is currently in use.

Prior to HP-UX 10.30

By default, the DPMS level used is the Screen Saver (i.e. no power savings). If you wish to use power saving during screen blanking, set the following **X*screens** file entry before starting the server:

`MinimumMonitorPowerSaveLevel <level>`

where *<level>* is replaced with the single digit 0, 1, 2, or 3 as specified in the **Level** column in the above table.

4-16 X Windows: HP-UX 10.x

HP-UX 10.30 and Beyond

The X Consortium proposed standard DPMS Extension has been implemented and is available on HP-UX 10.30, and will be available on all subsequent HP-UX releases.

The DPMS Extension lets individual users customize their personal DPMS settings to meet their work styles and any restrictions imposed by their employers. For example, an employer may decide that all monitors must save power after 30 minutes of idle time. The individual user may decide that 30 minutes is too long, and adjust the time downward to meet his or her own work preference.

More information (including sample code) on the DPMS Extension entrypoints can be found on-line, via man pages, or via HP's X Server Team homepage at:

<http://www.hp.com/go/xwindow>

The extension entrypoints are:

- DPMS
- DPMSQueryExtension
- DPMSGetVersion
- DPMSCapable
- DPMSSetTimeouts
- DPMSGetTimeouts
- DPMSEnable
- DPMSDisable
- DPMSForceLevel
- DPMSInfo

MBX

The MBX extension (Multi-Buffering Extension) is supported on all graphics devices supported on the HP 9000/700 machines, except the PersonalVRX and the TurboVRX.

HP's implementation of MBX exists mainly to support fast double-buffering for PEX applications. Therefore, MBX only supports allocation of one or two MBX buffers; no more. Some graphics devices/visuals have a single 8-plane buffer; this includes the color graphics device and the overlay planes on the CRX-24[Z], CRX-48Z, HCRX, and HP VISUALIZE family. For these devices, MBX double-buffering is still supported, but the second bank is allocated in virtual memory. Rendering and buffer-swapping in these instances is slower than devices/visuals that support true hardware double-buffering.

4 There is no easy way to determine which visuals, from a device's list of visuals, support fast MBX hardware double-buffering. The CRX and Dual-CRX device is a double-buffered device and therefore always supports MBX hardware double-buffering. The Internal Color Graphics, Integrated Color Graphics or Color Graphics card devices only support MBX software buffering. All other devices that have both overlay and image planes support fast MBX hardware double-buffering in the image planes and slower MBX software double-buffering in the overlays. Consult the following device-specific sections for a list of visuals that support software and hardware MBX double-buffering.

For performance reasons, the default MBX behavior is to *not* synchronize with the monitors vertical retrace period. In some instances, image tearing could be visible while swapping large MBX windows. For those instances where tearing would occur and is undesirable, an optional X server mode is available to allow for synchronization with vertical retrace. To activate this optional X server mode, set the `SwapBuffersOnVBlank` Screen Option in the `X*screens` file before the X server is started.

Note that `MBX_SWAP_BUFFERS_ON_VBLANK` is obsolete with this release. The `SwapBuffersOnVBlank` Screen Option works for both DBE and MBX.

With this mode enabled, all MBX buffer swaps are synchronized with the monitor's vertical retrace period.

This mode is not needed in drawables used for PEX rendering. PEX turns synchronization on and thus does not require this tuning.

4-18 X Windows: HP-UX 10.x

The MBX Application Programming Interface is thoroughly discussed in the *PEXlib Programming Manual* by Tom Gaskins, and published by O'Reilly & Associates, Inc. Consult that manual to understand the creation, manipulation, and destruction of MBX buffers.

Since MBX is not an industry standard, and will be discontinued on HP-UX 11.0, developers should replace MBX calls with the appropriate DBE calls.

Note

Note that `XmbufGetScreenInfo()` can indicate that a window supports MBX even if only one MBX buffer is supported. An application should always check the `max_buffers` field in the returned `XmbufBufferInfo` structure before assuming that a window supports two MBX buffers.

Shared Memory Extension (MIT_SHM)

The MIT shared memory extension provides both shared-memory XImages and shared-memory pixmaps based on the SysV shared memory primitives.

Shared memory XImages are essentially a version of the XImage interface where the actual image data is stored in a shared memory segment, and thus need not be moved through the Xlib interprocess communication channel. For large images, use of this facility can result in increased performance.

Shared memory pixmaps are a similar concept implemented for the pixmap interface. Shared memory pixmaps are two-dimensional arrays of pixels in a format specified by the X server, where the pixmap data is stored in the shared memory segment. In all other respects, shared memory pixmaps behave the same as ordinary pixmaps and can be modified by the usual Xlib routines. In addition, it is possible to change the contents of these pixmaps directly without the use of Xlib routines merely by modifying the pixmap data.

Supported Devices

The X server supports the MIT shared memory extension on the following devices:

- Internal Color Graphics
- Integrated Color Graphics
- CRX-24[Z]
- CRX-48Z
- HCRX-8[Z]
- HCRX-24[Z]
- HP VISUALIZE-EG
- HP VISUALIZE-8
- HP VISUALIZE-24
- HP VISUALIZE-48

4

Shared Memory Transport (SMT)

Shared Memory Transport (SMT) is a means to more rapidly transport large amounts of data from the client to the server. It is distinct from the MIT Shared Memory Extension, which is specifically for various types of images, although SMT can be used with that extension.

SMT is particularly advantageous for operations that move large amounts of data in a single request, such as a polyline or a polypoint, and for images when the MIT Shared Memory Extension is not used. It will work with the Big Requests Extension, but whether it will exhibit a performance increase depends on the size of the actual extended size request. There are some X requests for which no improvement is expected.

SMT is the default transport for 10.20 whenever a display name of any of the forms listed below are used, and when the client and server are actually on the same host. Note that “:0.0” is used for simplicity. This behavior is equally applicable for displays such as “:1.0”, etc.

- :0.0
- local:0.0
- *<hostname>*:0.0
- shmlink:0.0

A display name of the form `unix:0.0` will force the use of Unix Domain Sockets (UDS), which is identical to the local transport used before HP-UX 10.20.

4-20 X Windows: HP-UX 10.x

A display name of the form *nn.nn.nn.nn:0.0* (where *nn.nn.nn.nn* is an IP address) will force the use of Internet Sockets, which is the remote transport normally used, and which can be used locally. (This will be slow.)

It is possible that an application which violates the X interface standard will run correctly using UDS but hang or coredump when using SMT. Users encountering this problem can use:

```
DISPLAY=unix:0 <command_and_args>
```

to run the application compatibly, but without the performance improvement of SMT.

Note that if neither SMT nor UDS are desired, setting `XFORCE_INTERNET=True` before starting the X server forces all protocol to interact directly with the hardware internet card.

SMT uses file space on the file system containing `/var/spool/sockets/X11`. Should it be the case that that file system is full, the X server will use Unix Domain Sockets (UDS) but print a warning (in `/var/vue/Xerrors` if VUE is in use or `/var/dt/Xerrors` CDE is in use) on each connection startup. To address this (and if space cannot be made), `/var/spool/sockets/X11` can be a symbolic link to another file system with more space. If `/var/spool/sockets/X11` is on a NFS file system, currently SMT will (silently) not start, and the connection will be made using Unix Domain Sockets. Again, a symbolic link to a conventional file system may be used to deal with this.

4

Performance Tuning of SMT

The default values of some buffer sizes have been tuned for optimal performance in most situations. However, these are not necessarily optimal in all conditions. Under some circumstances system performance might be optimized (possibly at the expense of X performance) by tuning these parameters. Under most circumstances this should be unnecessary.

The Server accepts these parameters via the `X*screens` file, in the `ServerOptions` section. In this case, the default for all SMT connections is set.

The client accepts these parameters via the environment variable `X_SMT_SIZES`. For the client, the value affects all client connections to the server made while this environment variable is set.

In either case, the format and meaning of the fields is the same:

$\langle region_size \rangle$ [, $\langle high_water \rangle$ [, $\langle buffer_size \rangle$]]

with no embedded blanks. For examples:

```
32000,16000,5000
32000
0
```

The default is 100000,90000,90000.

The values are accepted as positive decimal, hex, or octal values according to the C conventions.

The special value of 0 (for buffer size; all other values are ignored) indicates that SMT is to be suppressed.

4

$\langle region_size \rangle$ controls the amount of shared memory allocated for the transport (in bytes). This has the largest effect on system performance. The value is rounded up to the next page boundary. Larger values yield faster X performance but there is a point of diminishing returns. The default is 100000 (which is rounded to 0x19000).

$\langle high_water \rangle$ is a soft boundary which affects the load on the Virtual Memory system. The value is rounded up to the next page boundary. The smaller the value, the smaller the number of pages actually used while sending “normal, small” X messages. Large messages can still be sent at high efficiency. In a memory-poor system making this small may be an advantage, but if sufficient memory is available, the value should be near the default.

The default value for $\langle high_water \rangle$ is 90000 or, if $\langle region_size \rangle$ is given, 1/8 of the region size (which is appropriate in memory-poor systems), with a minimum of 4096. If $\langle high_water \rangle$ is specified (and if $\langle region_size \rangle$ is specified, $\langle high_water \rangle$ usually should be also) it must be less than the $\langle region_size \rangle$.

$\langle buffer_size \rangle$ is the size used for the “small” requests that X normally generates protocol for. It is extremely unlikely that this will need tuning. It is not rounded. It must be at least 4096, but defaults to the

4-22 X Windows: HP-UX 10.x

same as the *⟨high_water⟩* size (if the option is used). Space is left for a control region if necessary.

The *⟨high_water⟩* value must fit within the region (and should be smaller), the buffer must fit within the high-water mark (and consequently the buffer must fit within the whole region).

If these parameters are used, be sure to confirm that they actually cause an improvement in actual usage situations. Incorrect values can degrade performance.

Note

Begin Note for Programmers:



X Applications which call `fork()`, and access the same display structure from both the parent and the child cannot be expected to operate reliably without extreme care (if at all), whether or not SMT is used. However, SMT is more sensitive to this than UDS. The problem is quite similar to `stdio`, where `fflush()` must be used to assure that data makes it from the buffer onto the file exactly once.

Similarly to `stdio`'s use of `fflush()`, `XFlush()` (not `_XFlush()`) must be called immediately before any `fork()` call that will be accessing the display from both the parent and child, as well as any time control is transferred between the parent and child, or vice-versa. (Calls to `fork()` which immediately do an `exec()` are not a problem.)

The SMT library code attempts to detect improper use of display connections after a fork, and issues a warning at runtime. However, not all all such usages can be detected. Symptoms include reporting the error, and applications hanging.

Also, because the parent and child might read from the same display connection (either replies or events) the library can detect inconsistent sequence numbers, which it will report. It will attempt to recover from such errors, but depending on what the application has done, recovery cannot always be successful.

Only for R5 Applications

SMT requires a change to an internal interface with the X library. In theory, no application should be calling this interface, but some applications, including at least one X test suite, are known to call it. The interface is `_XConnectDisplay`. Applications using it directly may not be able to use SMT for the display specified, and must add an extra (ninth) parameter, which is the address of an integer:

```
int dummy;

_XConnectDisplay(..., &dummy);
```

Symptoms include both damaged data and core dumps.

(There was an earlier HP Shared Memory Transport, which this one replaces. It used the same parameter, so it may be the case that any such calls have already been fixed.)

This problem does not occur in the R6 library.

End Note for Programmers

HP Color Recovery

Color Recovery is a technique that generates a better picture by eliminating the graininess caused by traditional dithering techniques. It is available on these graphics devices:

- Integrated Color Graphics and plug-in Color Graphics cards
- HCRX: HCRX-8[Z], HCRX-24[Z], HP VISUALIZE-EG, HP VISUALIZE-8, HP VISUALIZE-24, and HP VISUALIZE-48.

Color Recovery is available when using either depth-8 PseudoColor or depth-8 TrueColor visuals.

There are two components to the Color Recovery process. First, a different dither-cell size (16×2) is used when rendering shaded polygons. Second, a digital filter is used when displaying the contents of the frame buffer to the screen.

Under some conditions, Color Recovery can produce undesirable artifacts in the image (this also happens with dithering, but the artifacts are different). However,

4-24 X Windows: HP-UX 10.x

images rendered with Color Recovery are seldom worse than what dithering produces. In most cases, Color Recovery produces significantly better pictures than dithering.

Color Recovery is available by default for all depth-8 color visuals on devices that support the feature. If, for some reason, you wish to disable Color Recovery, set the `DisableColorRecovery` Screen Option in the `X*screens` file before starting the server (note that this disables Color Recovery for 3D APIs as well).

Color Recovery is enabled in conjunction with a particular X colormap that is associated with your window. If the X colormap is not installed in hardware, you may not see the effect of the Color Recovery filter (you may not even see the correct colors for that window). Given that more than one hardware colormap (or “color lookup table”) is available, this should happen infrequently.

The Color Recovery colormap is a *read-only* colormap. Any attempts to change it will be ignored and no error will be reported.

Access to the Color Recovery capability is transparent when using a 3D graphics API such as Starbase, HP-PHIGS or PEX. If you are producing graphics using Xlib calls, your application must perform some of the necessary processing. The method to access Color Recovery via Xlib is described in a section called “Accessing HP Color Recovery Technology via Xlib” in the device-dependent sections.

HP Color Recovery Extension

An extension has been added to the HP X server in HP-UX 10.30 that provides easy access to HP Color Recovery technology and HP’s dithering hardware from Xlib applications.

If an application currently has code to create, generate, and display 24-bit images, only a few slight modifications are necessary to utilize the HP Color Recovery Extension. The modifications include making sure the new extension (`HP-COLOR-RECOVERY`) exists, issuing a request to determine which colormap and/or visual should be used to create the 8-bit window with, and changing the reference to the `XPutImage` call.

Also, the HP Color Recovery extension is hardware independent! It doesn’t matter what graphics hardware is installed on the system. If the graphics hardware supports HP Color Recovery, it will be used. If the graphics hardware does not support HP Color Recovery and the hardware has dithering hardware,

X Windows: HP-UX 10.x 4-25

it is used to display the 24-bit image. If the hardware neither supports HP Color Recovery nor hardware dithering, software dithering is used.

The extension can even be used to software dither a 24-bit image to an 8-bit pixmap.

See the HP Color Recovery man pages, `HP-COLOR-RECOVERY(3x)`, for more information and coding examples. Additional information is also included in the next section.

Accessing HP Color Recovery Technology via the Color Recovery Extension

The HP Color Recovery Extension API has three entrypoints:

- `XhpCrQueryVersion(3x)`
- `XhpCrGetCmapAndVisual(3x)`
- `XhpCrPutImage(3x)`

Instead of using the method described below in the section entitled “Accessing HP Color Recovery Technology via Xlib,” the method described here is a nice alternative. The original method is worth reading, however, because it provides more insight into what is happening “behind the scenes” in this new extension.

Assuming that the client application already supports 24-bit rendering, this extension is a very simple to port.

First, if a 24-bit visual is not found on the target display and the HP Color Recovery extension exists, the client application should figure out which colormap and/or visual needs to be used to access HP Color Recovery. This is accomplished via a call to `XhpCrGetCmapAndVisual(3x)`. The returned colormap ID should be used to create the destination window and subsequent rendering requests.

Next, the 24-bit image is created as usual and filled with 24-bit data.

Finally, instead of calling `XPutImage` to display the data to the screen, the data is rendered using `XhpCrPutImage(3x)`. If the underlying hardware supports HP Color Recovery, it will be used to display the data. If Color Recovery is not supported on the underlying hardware, HP’s dithering hardware will be used to display the image. If neither are supported, a software dithering algorithm is applied to the data.

4-26 X Windows: HP-UX 10.x

The destination drawable of `XhpCrPutImage(3x)` may either be a Window or a Pixmap. If it is a Pixmap, the data will always be rendered using software dithering, regardless of what the underlying hardware supports.

Setting the environment variable `HP_DISABLE_COLOR_RECOVERY` can be used to control precisely which method is used to convert the 24-bit data to 8 bits. Constants are found in `hpcrP.h` which can be used to force a specific method. For example, `XhpCrMethodSwDithering` is currently defined to be 1. Setting `HP_DISABLE_COLOR_RECOVERY=1` will force the software dithering path.

The Color Recovery Extension will be ported to previous patch releases of HP-UX in the future. See <http://www.hp.com/go/xwindow> for more information on the Color Recovery Extension. Coding examples can be downloaded as well!

Dynamic Loading

HP's X server now dynamically loads the appropriate device drivers and extensions based on the target graphics display device and the extensions the device driver supports. This feature should be transparent to X server users.

When possible, the loading of X extensions is deferred until the first protocol request is encountered for a given extension. This feature should be transparent to X server users; however, it is expected to provide some performance enhancement.

Dynamically loaded modules are recorded by the X server in the files `"/var/X11/Xserver/logs/X*.log"`, where the "*" of `X*.log` reflects the display identifier for that given run. Only that last invocation against a given display identifier is retained. The log file contains the parsed contents of the given `X*screens` file and the full path name for all dynamically loaded modules for the given X server invocation. Deferred loaded modules are recorded as they are referenced.

Note



Altering or removing files under `/usr/lib/X11/Xserver` may prevent the X server from running.



Include Inferiors Fix

When a client application creates an X Graphics Context (GC), it is possible to specify the `subWindowMode` component. The two possible values are `ClipByChildren` (default) and `IncludeInferiors`. If the GC specifies `ClipByChildren`, any rendering to a window with inferior windows (i.e., the child is wholly enclosed by the parent) will appear only in the destination window. In other words, the rendering will not take place inside the inferiors. If the GC specifies `IncludeInferiors`, and the same rendering request is made, it is the responsibility of the X Server to ensure that the rendering is not clipped from the inferior windows. In other words, the rendering will appear in the destination window and the inferior windows.

4 With the advent of multi-layer devices, the `IncludeInferiors` mode became defective. Depending upon which layer or hardware buffer the destination drawable and inferior windows were in, the rendering may or may not have taken place. Also, the `GetImage` protocol clearly specifies that the default `GetImage` behavior is to include the depth-dependant contents of inferior windows (in other words, `GetImage` requires that `IncludeInferiors` work properly).

As of the 10.10 release, HP has offered a solution to the `IncludeInferiors` defect. Some customers create their test image archives using `XGetImage` (which currently returns incorrect data for multi-layer and double-buffered devices). Therefore, the Include Inferiors Fix will not be enabled by default. To enable the Include Inferiors Fix, add the `EnableIncludeInferiorsFix` Screen Option to the `X*screens` file.

For example:

```
Screen /dev/crt/  
    ScreenOptions  
        EnableIncludeInferiorsFix
```

This gives a system administrator control over when the fix is active and when it is not. In this way, each site can evaluate whether or not it is beneficial to enable this fix.

Shared Memory Usage With 3D Graphics

Graphics processes use shared memory to access data pertaining to the display device and X11 resources created by the server. (“Resources” includes windows, colormaps, and cursors.) The X11 server initiates an independent process called the Graphics Resource Manager (GRM) to manage these resources among graphics processes. Graphics processes include PEXlib, PHIGS, and Starbase applications. One problem encountered with GRM shared memory is that it may not be large enough to run some applications.

Graphics applications that require VM double-buffering use large amounts of shared memory. Shared memory can be completely consumed by several double-buffered graphics windows. When an application attempts to use more shared memory than is available, the application encounters errors and might terminate.

You can circumvent the problem by using Server Options to change the shared memory size.

Changing Graphics Shared Memory Size

The size of the shared memory segment used by the GRM can be controlled through a Server Option. The default value is 0x580000 (5.5 Mbytes) on Series 700 computers.

Note



The actual GRM shared memory size on a system can be determined by running “`ipcs -ma`”, finding the entry with CPID matching the process ID of the grmd process and then checking the segment size (SEGSZ) field.

If more shared memory space is needed, graphics shared memory size can be increased. For example, to set it to eight megabytes:

```
ServerOptions
GraphicsSharedMemorySize=0x800000
```

Note that the value must be in hexadecimal. The new value won’t take effect until you restart the X Server.

It is also possible to decrease the size of GRM shared memory. You may want to do this if you want to reduce the swap-space requirements of your system and/or

you do not intend to run any 3D graphics processes. For example, you could reduce graphics shared memory size to 0x100000 (one megabyte).

Count Transparent In Overlay Visual

In some configurations, an 8-plane overlay visual may have less than 256 colors. This should not cause a problem for most applications. If an application depends on 8-plane visuals having 256 colormap entries, this option may be useful. Setting this option will cause the X server to count transparent entries in the number of colormap entries.

Examples of Relevant Graphics Devices:

- CRX-24[Z], CRX-48Z, HCRX-8[Z], HCRX-24[Z], HP VISUALIZE-EG, HP VISUALIZE-8, HP VISUALIZE-24, and HP VISUALIZE-48.
- X*screens File Screen Option To Use: `CountTransparentInOverlayVisual`

Enable Overlay Transparency

This option is used to enable the usage of an overlay transparent color on devices that can support it, but, by default, do not allow it (for example, HCRX-8).

Examples of Relevant Graphics Device:

- HCRX-8[Z], HP VISUALIZE-EG, HP VISUALIZE-8
- X*screens File Screen Option To Use: `EnableOverlayTransparency`

3-Bit Center Color

This option is available to force the X server to center colors in the colormap to values that will reduce the amount of twinkle on flat-panel conversion. This option applies only to flat-panel displays.

The twinkling effect is caused by the analog-to-digital conversion. Due to noise in the analog signal, it is possible for a color near a boundary between two digital values to cause the conversion to bounce back-and-forth between the two colors (i.e., “twinkle”). In order to avoid this effect, the server “centers” the colors as far from the color boundaries as possible.

Examples of Relevant Graphics Device:

- Integrated Color Graphics, Color Graphics cards, Internal Color Graphics
X*screens File Screen Option To Use: `3BitCenterColor`

Image Text Via BitMap

When using the Xlib `XDrawImageString()` call to draw text, a visual effect may be seen where text appears to flicker as the background and foreground are drawn in distinct graphics operations. This option is available to eliminate the flicker effect but at the expense of reduced text performance. The option will make the X server first draw text to an off-screen pixmap prior to displaying it to the screen.

Examples of Relevant Graphics Device:

- Integrated Color Graphics, Color Graphics cards, CRX-24[Z], CRX-48Z, HCRX-8[Z], HCRX-24[Z], HP VISUALIZE-EG, HP VISUALIZE-8, HP VISUALIZE-24, and HP VISUALIZE-48
X*screens File Screen Option To Use: `ImageTextViaBitMap`

Note

Using this option will reduce text performance.



Obsolete Environment Variables

These HP-UX 9.x environment variables are no longer supported:

```
HP_SUPPRESS_TRUECOLOR_VISUAL
HP_COLORMAP_MANAGEMENT_SCHEME
WMSHMSPC
MBX_SWAP_BUFFERS_ON_VBLANK
CRX24_COUNT_TRANSPARENT_IN_OVERLAY_VISUAL
```

Special Device Files

Special device files are used to communicate between the computer and peripheral devices. The X server requires the use of a special device file for each graphics card present in the system. On HP-UX 10.x systems, five special graphics device files are automatically created. The first or primary graphics card, also known as the “console”, uses the “/dev/crt” or “/dev/crt0” device file. The others are called “crt1”, “crt2”, and “crt3” and also reside in “/dev”. Those systems containing multiple graphics devices on a single card (Dual Color Graphics and Dual CRX, for example) need to have special device files manually created for them.

Special device files are created with the “mknod” command. The `mknod` command resides in `/usr/sbin` and may only be invoked by a superuser (i.e., user `root`). Although special device files can be made in any directory of the HP-UX file system, the convention is to create them in the `/dev` directory. Any name may be used for the special device file; however, the names that are suggested for the devices are `crt`, `crt0`, `crt1`, `crt2`, or `crt3`. It is also acceptable to use a name that is descriptive of the graphics device, for example, `crt1.left` or `crt1.right`. See the manual page for `mknod(1M)` for more information regarding its usage.

All graphics special device files are character device files with read-write permissions by all. On 10.x systems, the major number will always be 174. The following table indicates which minor numbers to use for creating alternative device files. (The leading “0x” indicates that the number is in hexadecimal format.)

Table 4-2. Special Device Files on HP-UX 10.x

Device Filename	10.x Minor Number	Description
/dev/crt	0x000000	Standard console device file
/dev/crt.r	0x000000	Dual CRX Graphics console, right device
/dev/crt.l	0x000004	Dual CRX Graphics console, left device
/dev/crt1	0x010000	Secondary graphics device file
/dev/freedom	0x010000	Freedom Series, secondary graphics device file
/dev/crt1.r	0x010000	Secondary graphics device is Dual CRX Graphics, right device
/dev/crt1.l	0x010004	Secondary graphics device is Dual CRX Graphics, left device
/dev/crt2	0x020000	Third graphics device file
/dev/crt3	0x030000	Fourth graphics device file

4

Following are some examples of using the mknod entry for the HP-UX Operating System.

For example, an HP workstation running Dual CRX graphics, a sample 10.x mknod entry for the “left” graphics device would be:

```
/usr/sbin/mknod /dev/crt.l c 174 0x000004
chmod 666 /dev/crt.l
```

and the other, “right,” graphics device would be:

```
/usr/sbin/mknod /dev/crt.r c 174 0x010000  
chmod 666 /dev/crt.r
```

Note that once the device file has been created, it is necessary to ensure that it has read-write permissions by all; i.e. “chmod 666”.

Supported X Configurations

Supported Graphics Devices

The table below summarizes the graphics devices that are supported on each of the HP9000 Series systems.

Table 4-3. HP 9000 Supported Graphics Devices Table

Graphics Devices	Graphics Product Numbers	Supported HP 9000 Models
Integrated Grayscale Graphics*	N/A	712 (all models), 715/64, 715/80, 715/100, 725/100
Integrated Color Graphics with HP Color Recovery**	N/A	712 (all models), 715/64, 715/80, 715/100, 725/100, 748i/64, 748i/100, V743/64, V743/100
Color Graphics card with HP Color Recovery	A4077A	712 (all models), 715/64, 715/80, 715/100, 725/100, 748i/64, 748i/100, J200, J210, K100, K200, K210, K400, K410, D200, D210, D250, D310, D350
Dual Color Graphics card	A4078A	715/64, 715/80, 715/100, 725/100, J200, J210
Internal Grayscale Graphics*	N/A	705, 710, 715/33, 715/50, 715/75, 725/50, 725/75
Internal Color Graphics**	N/A	705, 710, 715/33, 715/50, 715/75, 725/50, 725/75, 745i/50, 745i/100, 747i/50, 747i/100
CRX, Dual CRX	A1659A, A2262A	720, 730, 735, 735/125, 750, 755, 755/125, 747i/50, 747i/100,
GRX	A4053A	720, 730, 735, 735/125
CRX-24	A2673A, A2271A, A2272A	715/33, 715/50, 715/75, 720, 725/50, 725/75, 730, 735, 735/125, 747i/50, 747i/100, 750, 755, 755/125

**Table 4-3.
HP 9000 Supported Graphics Devices Table (continued)**

Graphics Devices	Graphics Product Numbers	Supported HP 9000 Models
CRX-24Z	A2674A, A1454A	715/33, 715/50, 715/75, 720, 725/50, 725/75, 730, 735, 735/125, 750, 755, 755/125
CRX-48Z	A2675A	715/50, 715/75, 715/100, 725/50, 725/75, 725/100, 735, 735/125, 755, 755/125, J200, J210
HCRX-8, HCRX-8Z	A4070A, A4079A	715/64, 715/80, 715/100, 725/100, J200, J210
HP VISUALIZE-EG (internal)	N/A	C160L, C160U, C180-XP
HP VISUALIZE-EG card	A4450A	C160L, C160U, K460-XP
Dual HP VISUALIZE-EG	A4451A	C160L, C160U
HCRX-24, HCRX-24Z, HP VISUALIZE-8, HP VISUALIZE-24	A4071A, A4179A, A4441A, A4442A	715/64, 715/80, 715/100, 725/100, J200, J210
HP VISUALIZE-48	N/A	J200, J210
Freedom Series Graphics	A4091A, A4093A	715/80, 715/100, J200, J210
PersonalVRX, TurboVRX	(No longer on the Corporate Price List)	720, 730, 735, 735/125, 750, 755, 755/125

* Integrated Grayscale Graphics and Internal Grayscale Graphics is supported on high resolution (1280×1024) for all Models specified above.

** Integrated Color Graphics and Internal Color Graphics is supported on both medium-resolution (1024×768) and high-resolution (1280×1024) configurations of the Series 700 Models 705, 710, 712 (all models), and 715/33. High resolution is supported on all other Models specified above.

N/A Graphics product number not available at time of printing.

4-36 X Windows: HP-UX 10.x

Multi-Display Support

The following definitions are included to reduce confusion between the terms “multi-display,” “multi-screen,” “multi-seat,” and “single logical screen.”

Multi-Display A configuration with multiple graphics devices used concurrently. Any multi-screen, multi-seat, or single logical screen configuration is referred to as a multi-display configuration.

Multi-Screen A configuration in which a single X server with a mouse and keyboard drives multiple graphics devices (where each display is a different X Screen) concurrently while only allowing the cursor, not windows, to be moved between displays.

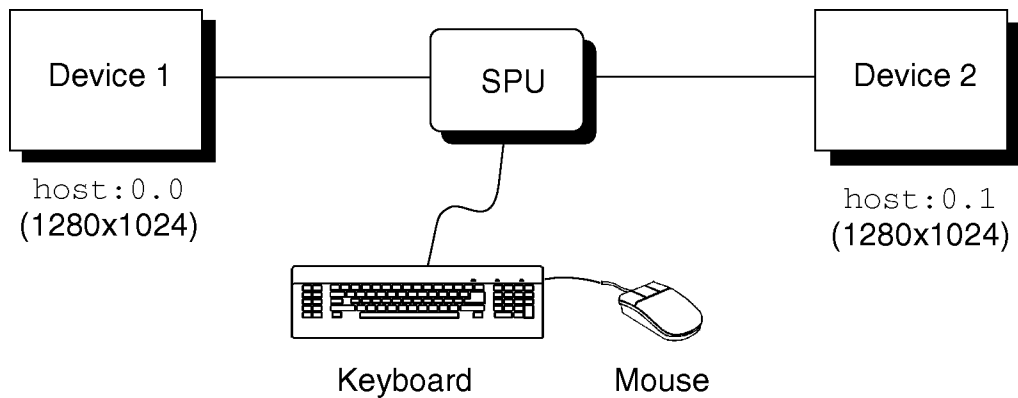


Figure 4-9.

Multi-Seat

A configuration with multiple instantiations of the X server, each with its own mouse, keyboard, and display(s). Multi-seat is not supported in any HP-UX 10.* release.

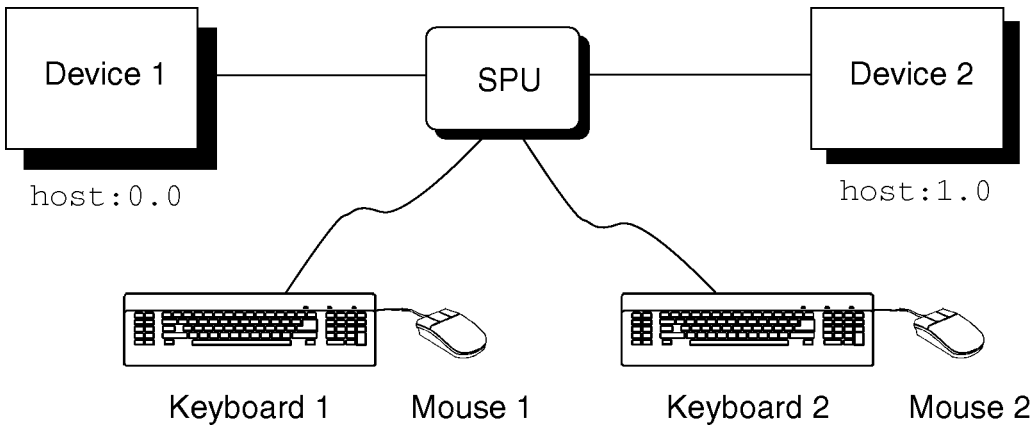
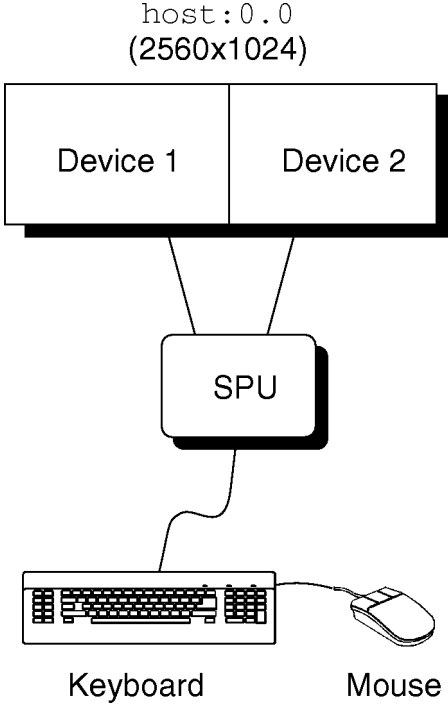


Figure 4-10.

Single Logical Screen A configuration in which a single X server with a single mouse and keyboard drives multiple homogeneous graphics devices concurrently while allowing the displays to emulate a large single screen. This differs from a multi-screen environment by allowing windows to be moved and displayed across displays. See the section in this document on Single Logical Screen.



4

Figure 4-11.

Note that different monitor resolutions are not supported with the multi-display configurations unless stated otherwise in the table below.

Multi-Screen Support

This section refers to multi-screen configurations only. Running one X server on more than one graphics display is called a “multi-screen” operation. The keyboard and pointer are shared among the screens. Multiple screens are enabled via the `/etc/X11/X*screens` file. The `X*screens` file is used to configure the operation of the X server. The screens are defined in the `X*screens` file by specifying the appropriate special device files. See the section in this document on special device files and `/etc/X11/X0screens` for more information.

A separate screen entry for each graphics display is entered in the `X*screens` file. The order of entries always determines the screen number starting at 0. The devices can be arranged in any order.

For example, in the following multi-screen system, the screen numbers are assigned as indicated:

```
Screen /dev/crt1      # first entry is screen 0 (as in local:0.0)
Screen /dev/crt0      # second entry is screen 1 (as in local:0.1)
Screen /dev/crt2      # third entry is screen 2 (as in local:0.2)
```

The X server supports up to four screens at a time. Specifying more than four screens will cause a server error message.

4

The following multi-screen configurations are supported:

- 712/60, 712/80, and 712/100:
 1. Integrated Color Graphics and one plug-in Color Graphics card.
- 715/33:
 1. Internal Color Graphics and one CRX-24.
 2. Internal Color Graphics and one CRX-24Z.
- 715/50, 715/75, 725/50, and 725/75:
 1. Internal Color Graphics and one CRX-24.
 2. Internal Color Graphics and one CRX-24Z.
 3. Internal Color Graphics and one CRX-48Z.
- 715/64, 715/80, and 715/100:
 1. Integrated Color Graphics and one plug-in Color Graphics card.
 2. Integrated Color Graphics and one Dual Color Graphics card.
 3. Integrated Color Graphics and one HCRX-8[Z].
 4. Integrated Color Graphics and one HP VISUALIZE-8.
 5. Integrated Color Graphics and one HCRX-24[Z].
 6. Integrated Color Graphics and one HP VISUALIZE-24.
 7. Integrated Color Graphics and one HP VISUALIZE-48.
- 725/100:
 1. Integrated Color Graphics and up to two plug-in Color Graphics cards.
 2. Integrated Color Graphics and one Dual Color Graphics card.
 3. Integrated Color Graphics and one HCRX-8[Z].
 4. Integrated Color Graphics and one HP VISUALIZE-8.
 5. Integrated Color Graphics and one HCRX-24[Z].
 6. Integrated Color Graphics and one HP VISUALIZE-24.
- 720, 730, 735, and 735/125:
 1. One Dual CRX.
- 748i/64 and 748i/100:
 1. Integrated Color Graphics and up to two plug-in Color Graphics cards (3×5).
 2. Integrated Color Graphics, one plug-in Color Graphics cards (3×5) and one HCRX-8 (3×5).
 3. Integrated Color Graphics or a plug-in Color Graphics cards (3×5) and one HCRX-8 (3×5).
 4. Integrated Color Graphics plug-in Color Graphics cards (3×5) and one HCRX-24 (3×5).
- 747i/50 and 747i/100:
 1. One Dual CRX.

2. Internal Color Graphics and one Dual CRX.

■ 750, 755 and 755/125:

1. One Dual CRX.
2. Two Dual CRXs.
3. Two CRX-24s.

■ C160L and C160U:

1. Built-In HP VISUALIZE-EG and one HP VISUALIZE-EG card.
2. Built-In HP VISUALIZE-EG and one HP VISUALIZE-24.
3. Built-In HP VISUALIZE-EG and one Dual HP VISUALIZE-EG card.
4. Built-In HP VISUALIZE-EG and one HP VISUALIZE-EG card and one Dual HP VISUALIZE-EG card.
5. Two HCRX-24 cards.

■ J200 and J210:

1. Up to four screens with any combination of plug-in Color Graphics cards and Dual Color Graphics cards.
 - a. One or two Dual Color Graphics cards.
 - b. Up to three plug-in Color Graphics cards.
 - c. One Dual Color Graphics card and up to two plug-in Color Graphics cards.
2. Two HCRX-24 cards.
3. One plug-in Color Graphics card and one HCRX-24Z.
4. One plug-in Color Graphics card and one HP VISUALIZE-24.
5. One plug-in Color Graphics card and one CRX-48Z.

■ K200, K210, K400, and K410:

1. Up to four plug-in Color Graphics cards.

■ K460:

1. Up to four plug-in HP VISUALIZE-EG cards.

* Unless otherwise stated, different resolutions are not supported in multi-display configurations.



Single Logical Screen (SLS)

SLS is a mechanism for treating homogeneous multi-display configurations as a single “logical” screen. This allows the moving/spanning of windows across multiple physical monitors. The word “homogeneous” is included because SLS only works if the graphics devices included in the SLS Configuration are of the same type. See the list of the supported SLS configurations shown below. SLS is enabled via the `/etc/X11/X*screens` file via the syntax:

```
SingleLogicalScreen n m
    /dev/crt0 ... /dev/crtk
```

where: n = the number of “rows” in the physical configuration, m = the number of “columns” in the physical configuration, and the product of $n \times m$ is less than or equal to four.

For example, to create a logical screen that is one monitor tall by two monitors wide, the following syntax would be used:

```
SingleLogicalScreen 1 2
    /dev/crt0 /dev/crt1
```

Whereas for a logical screen that is two monitors tall by one monitor wide, the syntax is:

```
SingleLogicalScreen 2 1
    /dev/crt0 /dev/crt1
```

Supported SLS Configurations

All supported multi-screen combinations (see the list of supported multi-screen configurations in the Multi-Display Support section) consisting of only one graphics device type (three Color Graphics devices, for example) are supported by SLS. A fairly comprehensive list of these configurations is shown below:

- Series 720 with a Dual CRX
- Series 712 with an Integrated Color Graphics + plug-in Color Graphics
- Series 715 with an Integrated Color Graphics + plug-in Color Graphics
- Series 725 with an Integrated Color Graphics + plug-in Color Graphics
- Series 730/735 with a Dual CRX
- Series 750/755 with a Dual CRX
- Series 750/755 with two Dual CRX plug-in cards
- Series 750/755 with two CRX-24 plug-in cards
- Series 770 (J-Series) with two HCRX-8 plug-in cards
- Series 770 (J-Series) with two HP VISUALIZE-8 plug-in cards
- Series 770 (J-Series) with two HCRX-24 plug-in cards
- Series 770 (J-Series) with a Dual Color Graphics card.
- C160L and C160U: All multi-screen configurations using only HP VISUALIZE-EG

3D Acceleration and Single Logical Screen

Currently, SLS does not take advantage of 3D acceleration (e.g. CRX-24Z). 3D applications (from any supported HP 3D API) will continue to run with SLS; However, 3D performance with SLS will be much slower than it is without SLS.

HP VUE/CDE and Single Logical Screen

Please note that HP VUE/CDE has not been modified to take advantage of the Single Logical Screen capability. When presenting information on your display, HP VUE may split a window across physical screens. Examples include:

- The login screen.
- The Front Panel.
- Window move and resize boxes.
- The screen lock dialog.

This behavior is the result of HP VUE's naive assumption that it is running against one large screen; it centers these windows accordingly.

4-44 X Windows: HP-UX 10.x

If you are using the default HP VUE key bindings, you can easily reposition the Front Panel so that it is completely contained within one physical screen:

1. With the input focus on the Front Panel, press **Alt Space** (on older keyboards, use **Extend Char Space**).
2. With the Front Panel menu posted and the “Move” menu item selected, press **Enter** (on older keyboards, **Return**) to start the move.
3. Use the mouse or the arrow keys to reposition the Front Panel to the desired location.
4. Press **Enter** (or **Return**) to complete the move. You may instead press **Esc** to cancel the move.

Afterwards, this setting will be remembered and restored at your next login. If you have previously set a Home session, you will need to re-set the Home session in the Style Manager to register the new Front Panel position.

Note that there is no mechanism in HP VUE for repositioning the login screen, window move/resize boxes, or the screen lock dialog.

4

Integrated Color Graphics Device-Dependent Information

This sections includes information on Integrated Color Graphics and Color Graphics cards.

Supported Visuals

For color displays:

- Class PseudoColor Depth 8—
supports DBE and MBX software double-buffering
- Class TrueColor Depth 8—
supports DBE and MBX software double-buffering

For grayscale displays, only one visual is supported:

- Class GrayScale Depth 8—
supports DBE and MBX software double-buffering

Supported Screen Options

The following Screen Options are supported:

- `DisableColorRecovery`
- `3BitCenterColor`
- `ImageTextViaBitMap`

Colormaps and Colormap Management

Color Graphics devices have two hardware colormaps (color lookup tables), each with 256 entries. The X server controls the allocation and contents of these hardware colormaps.

Default Colormap Management Scheme

Many applications use the default X11 colormap. A technicolor effect in the windows using the default colormap occurs when a non-default colormap is downloaded in the hardware colormap that had previously contained the default colormap.

Because so many applications use the default X11 colormap—including the window manager—and because Color Graphics devices have two hardware colormaps, the default behavior on this device is to dedicate one hardware colormap to always hold the default X11 colormap. The second hardware colormap is available to applications that use colormaps other than the default.

The default behavior can cause technicolor if two or more applications are using different, non-default colormaps. For example, Application A uses the default X11 colormap, Application B uses a different colormap, and Application C uses a third colormap. If applications A, B, and C are all executed simultaneously on a Model 712, application A would look correct. Either application B or C would have a technicolor effect—the application whose colormap was last downloaded in the hardware colormap would look correct.

Accessing HP Color Recovery Technology via Xlib

For HP-UX 10.30 and beyond, the HP Color Recovery Extension has been added to HP's X server. It provides a simple API for taking a 24-bit image and displaying it in an 8-bit window. See the section entitled

Accessing HP Color Recovery Technology via the Color Recovery Extension for more information.

Color Recovery is a technique to generate a better picture by attempting to eliminate the graininess caused by dithering. Access to the Color Recovery capability is transparent when using a 3D graphics API such as Starbase, HP-PHIGS or PEX. If you are producing graphics using Xlib calls, your application must perform some of the necessary processing. At server startup (if Color Recovery is not disabled in the `X*screens` file), the following properties are defined and placed on the root window:

- `_HP_RGB_SMOOTH_TRUE_MAP`
- `_HP_RGB_SMOOTH_PSEUDO_MAP`
- `_HP_RGB_SMOOTH_MAP_LIST`

These properties are of type `RGB_COLOR_MAP` and carry pointers to structures of type `XStandardColormap`. They may be interrogated with calls to `XGetRGBColormaps`. The colormaps in the `_HP_RGB_SMOOTH_TRUE_MAP` and `_HP_RGB_SMOOTH_PSEUDO_MAP` structures identify colormaps which are created at server startup and are for use with the TrueColor and PseudoColor visuals, respectively. They are both initialized to contain the 3:3:2 ramp of 8-bit TrueColor. Neither of these colormaps can be modified as they are *read-only*. The property `_HP_RGB_SMOOTH_MAP_LIST` is a list of colormaps that are associated with all of the root window's visual IDs that support Color Recovery. When the `XGetRGBColormaps` routine searches throughout this list for a colormap with a visual ID that matches the visual ID that your window is using and it finds one, your application knows that your visual supports Color Recovery, and uses that colormap for any Color Recovery window in your window's visual.

Note that the algorithm used for the Color Graphics device is slightly different from that used for the HCRX family of devices. If you do not wish for your application to have to do device-specific checks, HP recommends that you use the HCRX encoding algorithm for Color Recovery regardless of the device on which your application is executing. The results on the Color Graphics device will not be optimal, but will generally still be much better than a standard

dither. If you are willing to do device-specific checks, the existence of either the `_HP_RGB_SMOOTH_TRUE_MAP` or `_HP_RGB_SMOOTH_PSEUDO_MAP` property will indicate the device is Color Graphics.

Color Recovery uses all 256 entries of one of the available colormaps. The color visual used by Color Recovery emulates the 24-bit TrueColor visual; thus, the colors red, green, and blue are typically declared as integers in the range from 0 to 255. Note that each window that uses Color Recovery will have the same colormap contents.

For Color Recovery to produce the best results, the emulated 24-bit TrueColor data is dithered as explained below.

A pixel to be dithered is sent to the routine provided in this example. Note that the values of the variables `RedValue`, `GreenValue`, and `BlueValue` are generated by an application. In this example, the color values are assumed to be in the range 0..255.

The given routine receives the color values and the X and Y window address (X_p and Y_p) of the pixel. The X and Y address is used to access the dither tables. The values from the dither tables are added to the color values. After the dither addition, the resultant color values are quantized to three bits of red and green and two bits of blue. The quantized results are packed into an 8-bit unsigned char and then stored in the frame buffer. In the process of sending the contents of the frame buffer to the CRT, a special section in the hardware then converts the frame buffer's 8-bit data into 24-bit TrueColor data for display.

4

Here is a routine that can be used to dither the 24-bit TrueColor data.

```
unsigned char dither_pixel_for_CR(RedValue,GreenValue,BlueValue,Xp,Yp)
int   RedValue, GreenValue, BlueValue, Xp, Yp;
{
    static short    dither_red[2][16] = {
        {-16,  4, -1, 11,-14,  6, -3,  9,-15,  5, -2, 10,-13,  7, -4,  8},
        { 15, -5,  0,-12, 13, -7,  2,-10, 14, -6,  1,-11, 12, -8,  3, -9}};
    static short    dither_green[2][16] = {
        { 11,-15,  7, -3,  8,-14,  4, -2, 10,-16,  6, -4,  9,-13,  5, -1},
        {-12, 14, -8,  2, -9, 13, -5,  1,-11, 15, -7,  3,-10, 12, -6,  0}};
    static short    dither_blue[2][16] = {
        { -3,  9,-13,  7, -1, 11,-15,  5, -4,  8,-14,  6, -2, 10,-16,  4},
        {  2,-10, 12, -8,  0,-12, 14, -6,  3, -9, 13, -7,  1,-11, 15, -5} };
    int             red, green, blue;
    int             x_dither_table, y_dither_table;
    unsigned char   pixel;

    /* Determine the dither table entries to use based on the pixel address */
    x_dither_table = Xp % 16; /* X Pixel Address MOD 16 */
    y_dither_table = Yp % 2; /* Y Pixel Address MOD 2 */

    /* Start with the initial values as supplied by the calling routine */
    red  = RedValue;
    green = GreenValue;
    blue = BlueValue;

    /* Generate the red dither value */
    red += dither_red[y_dither_table][x_dither_table];
    /* Check for overflow or underflow on red value */
    if (red > 0xff) red = 0xff;
    if (red < 0x00) red = 0x00;

    /* Generate the green dither value */
    green += dither_green[y_dither_table][x_dither_table];
    /* Check for overflow or underflow on green value */
    if (green > 0xff) green = 0xff;
    if (green < 0x00) green = 0x00;

    /* Generate the blue dither value */
    blue += (dither_blue[y_dither_table][x_dither_table]<<1);
    /* Check for overflow or underflow on blue value */
    if (blue > 0xff) blue = 0xff;
    if (blue < 0x00) blue = 0x00;

    /* Generate the pixel value by "or"ing the values together */
    pixel = ((red & 0xE0) | ((green & 0xE0) >> 3) | ((blue & 0xC0) >> 6));
    return(pixel);
}
```

Internal Color Graphics, Internal Grayscale Graphics, CRX, GRX, and Dual-CRX Device-Dependent Information

Supported Visuals

Only one visual is supported.

For color displays:

- Class PseudoColor Depth 8—
 - supports DBE and MBX hardware double-buffering (CRX, Dual CRX)
 - supports DBE and MBX software double-buffering (Internal Color Graphics)

For grayscale displays:

- Class GrayScale Depth 8—
 - supports DBE and MBX hardware double-buffering (GRX)
 - supports DBE and MBX software double-buffering (Internal GrayScale Graphics)

The “layer” and “transparent” default visual options are not supported.

Supported Screen Options

The following Screen Options are supported:

- SwapBuffersOnVBlank
- 3BitCenterColor (Internal Color Graphics only)
- EnableIncludeInferiorsFix

CRX-24[Z] Device-Dependent Information

Supported Visuals

The following visuals are supported:

- Class PseudoColor Depth 8 Layer Image—
supports DBE and MBX hardware double-buffering
- Class PseudoColor Depth 8 Layer Overlay—
supports DBE and MBX software double-buffering
- Class DirectColor Depth 12 Layer Image—
supports DBE and MBX hardware double-buffering
- Class TrueColor Depth 12 Layer Image—
supports DBE and MBX hardware double-buffering
- Class DirectColor Depth 24 Layer Image—
doesn't support DBE and MBX double-buffering
- Class TrueColor Depth 24 Layer Image—
doesn't support DBE and MBX double-buffering

4



Supported Screen Options

The following Screen Options are supported:

- CountTransparentInOverlayVisual
- SwapBuffersOnVBlank
- ImageTextViaBitMap
- CRX24_FULL_DEFAULT_VISUAL
- EnableIncludeInferiorsFix

CRX-24[Z] Transparent Overlay Visuals

The default number of colormap entries in the overlay visual for the CRX-24[Z] is 255. Entry 255 is excluded because its value is hard-coded to transparent (that is, show the image planes).

This may have the following two consequences for X11 applications running in the overlay planes (the default visual):

- Clients attempting to allocate 256 entries do not have their request granted.
- Clients requesting (via `XAllocNamedColor`) the `rgb.txt` value of “Transparent” are not returned entry 255.

This default behavior can be changed by setting the `CountTransparentInOverlayVisual` screen option.

When this option is enabled, the X server does the following:

- Specifies that the overlay visual has 256 entries.
- Creates the default colormap with entry 255 pre-allocated to `Transparent`. A client calling `XAllocNamedColor` for entry `Transparent` in the default colormap will be returned entry 255.
- For all other colormaps, returns all 256 entries as allocable, but issues a warning message:

```
Warning: XCreateColormap is creating 256 entry cmaps in overlay visual.  
Though allocable, entry 255 is hard-coded to transparency.
```

This warning is issued once per server execution.

4

CRX-48Z Device-Dependent Information

Supported Visuals

The following visuals are supported:

- Class PseudoColor Depth 8 Layer Image—
supports DBE and MBX hardware double-buffering
- Class PseudoColor Depth 8 Layer Overlay—
supports DBE and MBX software double-buffering
- Class DirectColor Depth 24 Layer Image—
supports DBE and MBX hardware double-buffering
- Class TrueColor Depth 24 Layer Image—
supports DBE and MBX hardware double-buffering

Screen Options

The following Screen Options are supported:

- `CountTransparentInOverlayVisual`
- `SwapBuffersOnVBlank`
- `ImageTextViaBitMap`
- `EnableIncludeInferiorsFix`

CRX-48Z Transparent Overlay Visuals

The default number of colormap entries in the overlay visual for the CRX-48Z is 255. Entry 255 is excluded because its value is hard-coded to transparent (that is, show the image planes).

This may have the following two consequences for X11 applications running in the overlay planes (the default visual):

- Clients attempting to allocate 256 entries do not have their request granted.
- Clients requesting (via `XAllocNamedColor`) the `rgb.txt` value of `Transparent` are not returned entry 255.

This default behavior can be changed by setting the `CountTransparentInOverlayVisual` screen option.

When this option is enabled, the X server does the following:

- Specifies that the overlay visual has 256 entries.
- Creates the default colormap with entry 255 pre-allocated to `Transparent`. A client calling `XAllocNamedColor` for entry `Transparent` in the default colormap will be returned entry 255.
- For all other colormaps, returns all 256 entries as allocable, but issues a warning message:

```
Warning: XCreateColormap is creating 256 entry cmaps in overlay visual.  
Though allocable, entry 255 is hard-coded to transparency.
```

This warning is issued once per server execution.

4

HCRX and HP VISUALIZE Device-Dependent Information

This section includes information on the HCRX-8[Z], HCRX-24[Z], HP VISUALIZE-EG, HP VISUALIZE-8, HP VISUALIZE-24, and HP VISUALIZE-48 graphics devices.

The HCRX-8[Z] is a one board device (two, with the optional accelerator that has eight overlay planes, two banks of 8 image planes, and 4 hardware colormaps. This device provides a superset of functionality in the CRX.

The HCRX-24[Z] is a one board device (two, with the optional accelerator) that has eight overlay planes, two banks of 12 image planes, and 4 hardware colormaps. This device provides a superset of functionality in the CRX-24[Z].

The HP VISUALIZE-EG is either an unaccelerated built-in graphics device or a single board unaccelerated graphics device (not counting the optional memory daughter card in either case). This device provides compatible functionality with the Integrated Color Graphics device when in 8 plane mode and has functionality compatible with the HCRX-8 device when in double-buffer mode. See below for a description of these modes. For shorthand notation, from this point on in the document, HP VISUALIZE-EG will refer to either mode, HP VISUALIZE-EG(8) will refer to 8 plane mode only and HP VISUALIZE-EG(D) will refer to double-buffer mode only.

4-54 X Windows: HP-UX 10.x

The HP VISUALIZE-8 is a two board accelerated device that has eight overlay planes, two banks of 8 image planes, and 4 hardware colormaps. This device provides a superset of functionality in the CRX.

The HP VISUALIZE-24 is a two board accelerated device that has eight overlay planes, two banks of 12 image planes, and 4 hardware colormaps. This device provides a superset of functionality in the CRX-24[Z].

The HP VISUALIZE-48 is a two-board accelerated device (three, with the optional texture-mapping hardware) that has eight overlay planes, two banks of 24 image planes, and six hardware colormaps. This device provides a superset of functionality in the CRX-48Z. The hardware support for accelerating 2D Xlib primitives is similar to that in the other HCRX devices. The hardware for accelerating 3D geometry, lighting, and shading, is new.

Supported Visuals

The following visuals are supported on the HP VISUALIZE-EG(8):

- Class PseudoColor Depth 8 Layer Image—
supports DBE and MBX software double-buffering
- Class TrueColor Depth 8 Layer Image—
supports DBE and MBX software double-buffering

The following visuals are supported on the HCRX-8[Z], HP VISUALIZE-EG(D) and HP VISUALIZE-8:

- Class PseudoColor Depth 8 Layer Image—
supports DBE and MBX hardware double-buffering
- Class PseudoColor Depth 8 Layer Overlay - (see Note)
supports DBE and MBX software double-buffering
- Class PseudoColor Depth 8 Layer Overlay Transparent - (see Note)
supports DBE and MBX software double-buffering
- Class TrueColor Depth 8 Layer Image—
supports DBE and MBX hardware double-buffering

Note

The two overlay visuals are mutually exclusive, based on the presence of the `EnableOverlayTransparency` screen option (i.e., if the `EnableOverlayTransparency` screen option is set, then the visual that supports transparency *is* available, otherwise the visual which does *not* support transparency is available).

The following visuals are supported on the HCRX-24[Z] and HP VISUALIZE-24:

- Class PseudoColor Depth 8 Layer Image— supports DBE and MBX hardware double-buffering
- Class PseudoColor Depth 8 Layer Overlay— supports DBE and MBX software double-buffering
- Class PseudoColor Depth 8 Layer Overlay Transparent— supports DBE and MBX software double-buffering
- Class TrueColor Depth 8 Layer Image— supports DBE and MBX hardware double-buffering
- Class DirectColor Depth 12 Layer Image— supports DBE and MBX hardware double-buffering
- Class TrueColor Depth 12 Layer Image— supports DBE and MBX hardware double-buffering
- Class DirectColor Depth 24 Layer Image— doesn't support DBE and MBX double-buffering
- Class TrueColor Depth 24 Layer Image— doesn't support DBE and MBX double-buffering

The following visuals are supported on the HP VISUALIZE-48:

- Class PseudoColor Depth 8 Layer Image— supports DBE and MBX hardware double-buffering
- Class PseudoColor Depth 8 Layer Overlay— supports DBE and MBX software double-buffering
- Class PseudoColor Depth 8 Layer Overlay Transparent— supports DBE and MBX software double-buffering
- Class TrueColor Depth 8 Layer Image— supports DBE and MBX hardware double-buffering
- Class DirectColor Depth 24 Layer Image— supports DBE and MBX hardware double-buffering
- Class TrueColor Depth 24 Layer Image— supports DBE and MBX hardware double-buffering

4-56 X Windows: HP-UX 10.x

Supported Screen Options

The following Screen Options are supported:

- `CountTransparentInOverlayVisual`
- `DisableColorRecovery`
- `EnableOverlayTransparency` (HCRX-8[Z], HP VISUALIZE-EG(D) and HP VISUALIZE-8 only)
- `SwapBuffersOnVBlank`
- `ImageTextViaBitMap`
- `CRX24_FULL_DEFAULT_VISUAL` (HCRX-24[Z] only)
- `EnableIncludeInferiorsFix`

HP VISUALIZE-EG Modes

The following modes are supported:

- 8 Plane mode
- Double-Buffer mode

The modes are set from the Boot-Admin at bootup time by selecting from the menu of options a configuration that supports double-buffer or not. From that point on (without rebooting) the server will use the selected mode.

Eight-plane mode is compatible with the Integrated Color Graphics device. It has eight image planes and uses only software double-buffering.

Double-Buffer mode is compatible with the HCRX-8 device. This mode requires an optional memory daughter card. If the daughter card is installed, selecting this mode will result in eight overlay planes and 16 image planes (the same as HCRX-8 and HP VISUALIZE-8 devices). Double-Buffer mode allows the use of hardware double-buffering.

HCRX Configuration Hints

HCRX-8[Z], HP VISUALIZE-EG(D) and HP VISUALIZE-8 Visuals and Double-Buffer Support

The eight-plane HCRX-8[Z], HP VISUALIZE-EG(D) and HP VISUALIZE-8 are the first members of the Series 700 graphics family whose overlay planes and image planes are both depth 8.

- There are two depth-8 PseudoColor visuals (one in the overlay planes, the other in the image planes). There is also a depth-8 TrueColor visual in the image planes.
- The default visual (where the root window and default colormap reside) is in the overlay planes. A `DefaultVisual` specification in a Screen Entry in the `X*screens` file may instead locate the default visual in the Image Planes (see the `X*screens` File section, above).
- Fast 8/8 double-buffering (two hardware buffers) is supported in the depth-8 image planes, but not in the overlays. The overlay planes support the slower virtual-memory-based double-buffering.

Implications and Suggestions for HCRX-8[Z], HP VISUALIZE-EG(D) and HP VISUALIZE-8

The default colormap cannot be used with a window in a non-default visual, even one of the same depth as the default visual.

Before trying to use the default colormap in a depth-8 window, verify that the window is in the default visual. If the window is not in the default visual, create a colormap in that visual. This process of creating a non-default colormap is the same as the one used to create windows in depth-12 or depth-24 visuals.

If you have an application that assumes that the default colormap can be used with *any* depth-8 window (even one in an image-plane visual) specify an image plane visual as the default.

Unlike the CRX, the HCRX-8[Z]'s default visual the HP VISUALIZE-EG(D)'s default visual and the HP VISUALIZE-8's default visual do not have fast hardware double-buffering (but the image planes do).

To obtain hardware double-buffering, find a visual in the image planes. The best method is to find all the depth-8 PseudoColor visuals returned by

4-58 X Windows: HP-UX 10.x

XGetVisualInfo and then eliminate the visuals that are reported in the SERVER_OVERLAY_VISUALS property (discussed below).

If you have an application that assumes the default visual has fast double-buffering, specify an image plane visual as the default.

HCRX Overlay Visuals and Overlay Transparency

As on the CRX-24[Z] and CRX-48Z, a property on the root window, SERVER_OVERLAY_VISUALS, is used to describe the visuals that are in the overlay planes.

Overlay Transparency on the HCRX-8[Z], HP VISUALIZE-EG(D) and HP VISUALIZE-8

The HCRX-8[Z], HP VISUALIZE-EG(D) and HP VISUALIZE-8 each have one visual in the overlay planes (depth-8 PseudoColor). By default, this overlay visual has no transparent index available to applications for rendering transparency. This means the overlay windows with “floating text” are not supported in the typical X server operation on the HCRX-8[Z], HP VISUALIZE-EG(D) or HP VISUALIZE-8.

For applications that require transparent overlay windows on the HCRX-8[Z], HP VISUALIZE-EG(D) or HP VISUALIZE-8, an optional X server mode is available to allow for overlay transparency, but it is restrictive. In this optional mode, overlay colormaps provide a single entry that can be used to render transparency. Only one hardware colormap is available in the overlays (instead of two) and only one hardware colormap is available in the image planes (instead of two).

To activate this optional X server mode to enable transparency, set the **EnableOverlayTransparency** screen option. You will need to restart the X server for the option to take effect.

With this mode enabled, colormaps created in the default visual have 255 entries; entry 256 is reserved for transparency. As on the CRX-24[Z] and CRX-48Z, the screen option **EnableOverlayTransparency** can be used to include the transparent index in the colormap size (256 entries instead of 255).



Note

For Programmers:

If transparency is not enabled, there are only 252 colors available. Entries 252-255 are not writable, and should not be used; there are only 252 colormap entries available, even though the server states that there are 256.

Overlay Transparency on the HCRX-24[Z], HP VISUALIZE-24, and HP VISUALIZE-48

The HCRX-24[Z], HP VISUALIZE-24, and HP VISUALIZE-48 have two visuals in the overlay planes, both depth-8 PseudoColor.

The default overlay visual has 256 entries per colormap and no transparency.

The second overlay visual has 255 entries per colormap and supports transparency in the same way as the CRX-24[Z]. As on the CRX-24[Z] and CRX-48Z, the screen option `EnableOverlayTransparency` can be used to include the transparent index in the colormap size (256 entries instead of 255).

To allow applications to determine which visuals are in the overlay planes, both overlay visuals are listed in the `SERVER_OVERLAY_VISUALS` property attached to the root window. The default overlay visual has a transparent type of 0 (`None`) while the transparent overlay visual has a transparent type of 1 (`TransparentPixel`).

4

If you need an overlay colormap that supports transparency, create the colormap using the visual that has transparency in its `SERVER_OVERLAY_VISUALS` property. To look at the contents of this property, you would use code similar to the following:

```

{
typedef struct {
    VisualID    overlayVisualID;
    Card32     transparentType; /* None, TransparentPixel, TransparentMask */
    Card32     value;          /* Either pixel value or pixel mask */
    Card32     layer;
} OverlayVisualPropertyRec;

OverlayVisualPropertyRec *pOverlayVisuals, *p0Vis;
XVisualInfo              getVis;
XVisualInfo              *pVisuals;
Atom                     overlayVisualsAtom, actualType;
...
/* Get the visuals for this screen and allocate. */
getVis.screen = screen;
pVisuals = XGetVisualInfo(display, VisualScreenMask, &getVis, &nVisuals);
pOverlayVisuals = (OverlayVisualPropertyRec *)
    malloc ( (size_t)nVisuals * sizeof(OverlayVisualPropertyRec) );

/* Get the overlay visual information for this screen. Obtain
 * this information from the SERVER_OVERLAY_VISUALS property. */
overlayVisualsAtom = XInternAtom(display, "SERVER_OVERLAY_VISUALS", True);
if (overlayVisualsAtom != None)
    {
    /* Since the Atom exists, request the property's contents. */
    bytesAfter = 0;
    numLongs = ( nVisuals * sizeof(OverlayVisualPropertyRec) + 3 ) / 4;
    XGetWindowProperty(display, RootWindow(display, screen),
        overlayVisualsAtom, 0, numLongs, False,
        AnyPropertyType, &actualType, &actualFormat,
        &numLongs, &bytesAfter, &pOverlayVisuals);
    if ( bytesAfter != 0 ) { /* Serious Failure Here */ } ;

    /* Loop through the pOverlayVisuals array. */
    ...
    n0Visuals = numLongs/sizeof(OverlayVisualPropertyRec);
    p0Vis = pOverlayVisuals;
    while (--n0Visuals >= 0)
        {
        if ( p0Vis->transparentType == TransparentPixel )
            { /* Found a transparent overlay visual, set ident. aside. */};
        p0Vis++;
        }
    XFree(pOverlayVisuals);
}

```

```

        /* There might be some additional checking of the found
           transparent overlay visuals wanted; e.g., for depth. */
    }
    XFree(pVisuals);
}

```

This program fragment is not complete; its main purpose is to give the idea of how to find an overlay visual having transparency.

HCRX Colormaps

The following information discusses the number of supported colormaps for the HCRX configurations.

HP VISUALIZE-EG(8): 8 Image planes

The image planes contain the default colormap permanently installed in the hardware plus one other hardware colormap available to applications. No issues involving transparency exist because of the lack of Overlay planes.

HCRX-8[Z], HP VISUALIZE-EG(D) and HP VISUALIZE-8: Eight Overlay Planes and Two Depth-8 Banks of Image Planes

When the default visual is in the overlay planes (default location) and the screen option `EnableOverlayTransparency` is not set, the overlay planes contain the default colormap permanently installed in the hardware, plus one other hardware colormap available to applications. The image planes contain two hardware colormaps each usable by applications.

When the default visual is in the image planes and the screen option `EnableOverlayTransparency` is not set, the overlay planes contain a single hardware colormap available to applications, plus a colormap reserved by the server (i.e., unavailable to applications) to guarantee the existence of transparency, and the image planes contain the default colormap permanently installed into the hardware, plus one other hardware colormap available to applications.

When the screen option `EnableOverlayTransparency` is set, both the overlay planes and the image planes have access to one hardware colormap. The default colormap is not permanently installed in the hardware and is in the overlay planes by default, but the Default Visual can be located in the image planes as described in a previous section.

4-62 X Windows: HP-UX 10.x

HCRX-24[Z] and HP VISUALIZE-24: Eight Overlay Planes and 24 Image Planes

The overlay planes contain the default colormap permanently installed in the hardware, plus one other hardware colormap available to applications. The image planes contain two hardware colormaps, each usable by applications.

Although two hardware colormaps are available to applications in the image planes, a hardware restriction allows only one depth-12 or depth-24 colormap to be installed at any given time. Therefore, if two applications are run simultaneously and use different depth-12 or depth-24 colormaps, the application that has the colormap focus looks correct and the other is technicolored.

HP VISUALIZE-48: Eight Overlay Planes and 48 Image Planes

The overlay planes contain the default colormap permanently installed in the hardware, plus one other hardware colormap available to applications. The image planes contain four hardware colormaps, each usable by applications.

The four hardware colormaps in the image planes can be treated as depth-8 or depth-24 colormaps. There are no restrictions on the types of colormaps that can be installed in the hardware at any given time. All four colormaps can be used with any visual class.

Accessing HP Color Recovery Technology via Xlib

For HP-UX 10.30 and beyond, the HP Color Recovery Extension has been added to HP's X server. It provides a simple API for taking a 24-bit image and displaying it in an 8-bit window. See the section entitled

Accessing HP Color Recovery Technology via the Color Recovery Extension for more information.

Color Recovery is a technique to generate a better picture by attempting to eliminate the graininess caused by dithering. Access to the Color Recovery capability is transparent when using a 3D graphics API such as Starbase, HP-PHIGS or PEX. If you are producing graphics using Xlib calls, your application must perform some of the necessary processing. At server startup (if Color Recovery is not disabled in the `X*screens` file), the `_HP_RGB_SMOOTH_MAP_LIST` property is defined and placed on the root window.

The above property is of type `RGB_COLOR_MAP` and carries pointers to structures of type `XStandardColormap`. It may be interrogated with calls to `XGetRGBCol-`

X Windows: HP-UX 10.x 4-63

ormaps. The property `_HP_RGB_SMOOTH_MAP_LIST` is a list of colormaps that are associated with window visual IDs that support Color Recovery. When the `XGetRGBColormaps` routine searches throughout this list for a colormap with a visual ID that matches your window's visual ID and it finds one, your application knows that your visual supports Color Recovery, and uses that colormap for any Color Recovery window in your window's visual.

Color Recovery uses all 256 entries of one of the available colormaps. The color visual used by Color Recovery emulates the 24-bit TrueColor visual, thus, the colors red, green, and blue are typically declared as integers in the range from 0 to 255. Note that each window that uses Color Recovery will have the same colormap contents.

For Color Recovery to produce the best results, the emulated 24-bit TrueColor data is dithered as explained below.

4

A pixel to be dithered is sent to the routine provided in this example. Note that the values of the variables `RedValue`, `GreenValue`, and `BlueValue` are generated by an application. In this example, the color values are assumed to be in the range 0..255.

The given routine receives the color values and the X and Y window address (X_p and Y_p) of the pixel. The X and Y address is used to access the dither tables. The values from the dither tables are added to the color values. After the dither addition, the resultant color values are quantized to three bits of red and green and two bits of blue. The quantized results are packed into an 8-bit unsigned char and then stored in the frame buffer. In the process of sending the contents of the frame buffer to the CRT, a special section in the hardware then converts the frame buffer's 8-bit data into a 24-bit TrueColor data for display.

Here is a routine that can be used to dither the 24-bit TrueColor data.

```
unsigned char dither_pixel_for_CR(RedValue,GreenValue,BlueValue,Xp,Yp)
int          RedValue,GreenValueBlueValue,Xp,Yp;
{
    static short    dither_red[2][16] = {
        {-16,  4, -1, 11,-14,  6, -3,  9,-15,  5, -2, 10,-13,  7, -4,  8},
        { 15, -5,  0,-12, 13, -7,  2,-10, 14, -6,  1,-11, 12, -8,  3, -9}};
    static short    dither_green[2][16] = {
        { 11,-15,  7, -3,  8,-14,  4, -2, 10,-16,  6, -4,  9,-13,  5, -1},
        {-12, 14, -8,  2, -9, 13, -5,  1,-11, 15, -7,  3,-10, 12, -6,  0}};
    static short    dither_blue[2][16] = {
        { -3,  9,-13,  7, -1, 11,-15,  5, -4,  8,-14,  6, -2, 10,-16,  4},
        {  2,-10, 12, -8,  0,-12, 14, -6,  3, -9, 13, -7,  1,-11, 15, -5}};
    int             red, green, blue;
    int             x_dither_table, y_dither_table;
    unsigned char   pixel;

    /* Determine the dither table entries to use based on the pixel address */
    x_dither_table = Xp % 16; /* X Pixel Address MOD 16 */
    y_dither_table = Yp % 2; /* Y Pixel Address MOD 2 */

    /* Start with the initial values as supplied by the calling routine */
    red  = RedValue;
    green = GreenValue;
    blue = BlueValue;

    /* Generate the red dither value */
    if (red >= 48) /* 48 is a constant required by this routine */
        red=red-16;
    else
        red=red/2+8;
    red += dither_red[y_dither_table][x_dither_table];
    /* Check for overflow or underflow on red value */
    if (red > 0xff) red = 0xff;
    if (red < 0x00) red = 0x00;

    /* Generate the green dither value */
    if (green >= 48) /* 48 is a constant required by this routine */
        green=green-16;
    else
        green=green/2+8;
    green += dither_green[y_dither_table][x_dither_table];
    /* Check for overflow or underflow on green value */
    if (green > 0xff) green = 0xff;
    if (green < 0x00) green = 0x00;
}
```

```

/* Generate the blue dither value */
if (blue >= 112) /* 112 is a constant required by this routine */
    blue=blue-32;
else
    blue=blue/2+24;
blue += (dither_blue[y_dither_table][x_dither_table]<<1);
/* Check for overflow or underflow on blue value */
if (blue > 0xff) blue = 0xff;
if (blue < 0x00) blue = 0x00;

pixel = ((red & 0xE0) | ((green & 0xE0) >> 3) | ((blue & 0xC0) >> 6));
return(pixel);
}

```

Freedom Series Graphics Device-Dependent Information

This sections describes support for the Freedom Series from Evans & Sutherland on Hewlett-Packard workstations.

Note



Note that the Freedom Series is no longer supported as of HP-UX 10.30; the information below is presented for those who are running previous versions of the operating system.

Supported Visuals

The following visuals are supported:

- Class PseudoColor Depth 8 Layer Overlay—
Class PseudoColor Depth 8 Layer Image—
supports DBE and MBX hardware double-buffering
- Class DirectColor Depth 24 Layer Image—
supports DBE and MBX hardware double-buffering
- Class TrueColor Depth 24 Layer Image—
supports DBE and MBX hardware double-buffering

Supported Screen Options

The following Screen Options are supported:

- `FreedomVideoFormat`

Freedom Video Formats

Freedom Series graphics devices have the ability to support several different video formats. The default format is 1280×1024 @ 75 Hz VESA timing. Other supported video formats may be selected by using the `FreedomVideoFormat` screen option in the appropriate `X*screens` file. This screen option replaces the 9.07 environment variable, `ES_VIDEO_FORMAT`. The appropriate video format must be selected to support the specific display device connected to the Freedom accelerator. Multisync monitors can support several different video formats.

Alternative supported formats:

4

Table 4-4. Alternative Supported Freedom Video Formats

Screen Option	Resolution	Description
<code>ntsc_cvo</code>	640×480	U.S. composite TV format with CVO*
<code>pal_cvo</code>	768×576	European composite TV format with CVO*

* “CVO” is the Composite Video Output card, which must be installed in the Freedom accelerator for this video format to work.

While the following formats are provided there is no intent to claim “support” for these formats, they are untested and unsupported configurations.

Alternative unsupported formats:

Table 4-5. Alternative Unsupported Freedom Video Formats

Screen Option	Resolution	Description
ntsc	640×480	U.S. composite TV format
pal	768×576	European composite TV format
hdtv	1920×1024	60 Hz interlaced
stereo1	640×512	60 Hz frame rate per eye
stereo2	1280×512	60 Hz frame rate per eye
vga	640×480	Standard VGA
video60	1280×1024	60 Hz
video76	1280×1024	76 Hz

4

VRX Device-Dependent Information

This section includes information on the PersonalVRX (PVRX) and TurboVRX (TVRX) graphics devices.

Note



Note that the PersonalVRX and the TurboVRX are no longer supported as of HP-UX 10.30; the information below is presented for those who are running previous versions of the operating system.

Supported Visuals

The following visuals are supported:

- Depth 3 (overlay and combined mode)
- Depth 4 (overlay and combined mode)
- Depth 8 (image and combined mode)
- Depth 12 (image and combined mode, TVRX only)
- Depth 16 (Creates a double-buffer version of the Depth 8 visual)
- Depth 24 (image and combined mode, TVRX only)

4-68 X Windows: HP-UX 10.x

None of these visuals support DBE and MBX double-buffering.

In image mode the default visual is the Depth 8 PseudoColor visual. In overlay mode it is the depth 3 or depth 4 PseudoColor visual as specified by the device file. In combined mode the first device file specifies the default visual. Examples are shown in the section below.

VRX Device Files

Different device files exist for the image planes and overlay planes on VRX devices. The following table shows examples of device files for VRX devices:

Table 4-6. VRX Device Files

Device Filename	10.x Major Number	10.x Minor Number	Description
/dev/crt	174	0x000000	Image mode
/dev/ocrt	174	0x000001	Overlay mode (3 planes)
/dev/o4crt	174	0x000003	Overlay mode (4 planes)

4

The X server supports three different modes of operation on VRX devices: image, overlay or combined.

In image mode, the X server runs only in the image planes. This is the default on VRX devices. To operate in image mode, the image device file should be specified as the primary screen device. For example:

```
/dev/crt # Image mode
```

In overlay mode, the X server runs only in the overlay planes. Since only 3 or 4 planes are available in the overlay planes on VRX devices, the number of colors is very limited. To operate in overlay mode, the overlay device file should be specified as the primary screen device. For example:

```
/dev/ocrt # Overlay mode using 3 overlay planes  
or  
/dev/o4crt # Overlay mode using 4 overlay planes
```

X Windows: HP-UX 10.x 4-69

In combined mode, the X server runs in both image and overlay planes. To configure the X server to operate in combined mode, a primary and a secondary device must be specified. The `VRXSecondaryDevice` is used for this purpose. For example:

```
/dev/ocrt /dev/crt          # default visual lives in overlay planes
      or
/dev/crt /dev/ocrt         # default visual lives in image planes
```

X Windows Configuration Details

This chapter discusses several details concerning the configuration of X hosts, colormaps, mouse, and keyboard.

Making an X*.hosts File

The `/etc/X0.hosts` file is an ASCII text file containing the hostnames of each remote host permitted to access your local server.

- If you are running as a stand-alone system, you must have your system's name in this file.
- If you are part of a network, the other system names must be included.

The syntax is as follows:

```
<host>  
<host>  
<host>
```

For example, if you are `hpaaaaa`, and regularly ran clients on `hpccccc`, and `hpdddd`, you would want the following lines.

```
hpaaaaa  
hpccccc  
hpdddd
```

Note that aliases work as well as hostnames, provided they are valid, that is, commonly known across the network.

X0.hosts and X0screens Relation

The default screen configuration file `X0screens` uses the default X11 remote host file `X0.hosts`.

Each custom `X*screens` file is associated with a special `X*.hosts` file. The number represented by the “*” causes the correct screen and host files to be used together. For example, `X3screens` takes an `X3.hosts` file. Both are referenced by the server when it is started with a `/usr/bin/X11/X :3` command.

If you use a special `X*screens` file, you need to set your `DISPLAY` variable appropriately. For the previous example, it would be set to `hostname:3.0`.

Note



The number in an `Xnscreens` file does not necessarily refer to a physical screen number; any meaning implied by the number is for the user to define. There are no semantics applied to the number except that the `Xnscreens` files are used when X is started on display `<name>:n.0`. For example, an `X3screens` file does not necessarily imply device file `/dev/crt3`; an `X3screens` file can use whatever device file the user specifies. The same applies to the `X*devices`, `X*.hosts`, `X*.pointerkeys`, etc., files as well.

5

Using an `/etc/hosts` File

This file need not be present if your system is configured to query a nameserver.

The `/etc/hosts` file is an ASCII text file containing a list of all the host names and internet addresses known to your system, including your own system.

If your system is not connected to a network, use the loopback address (`127.0.0.1`) and the hostname `unknown`:

```
127.0.0.1 unknown
```

For a local system to access a remote host:

- The address and hostname of the remote host must be listed in the local system's `/etc/hosts` file.
- The user must have a valid login (username and password) and home directory on the remote host.

5-2 X Windows Configuration Details

Using Special Input Devices

Input devices are connected to Hewlett-Packard computers through several different hardware interfaces. Among the interfaces supported are the Hewlett-Packard Human Interface Link (HP-HIL) and the industry standard RS-232C (serial) and DIN interfaces. Some Hewlett-Packard computers do not support all of these interfaces.

How the X Server Chooses the Default Keyboard and Pointer

The X server can access input devices through any of the above interfaces. Devices that use the HP-HIL interface and devices that use the DIN interface and are compatible with the HP DIN keyboard and mouse can be used by simply plugging them into the computer. Devices that use the RS-232C interface require the installation of input device driver software before they can be used.

If no explicit input device configuration is done, the X server chooses the X keyboard device and X pointer device from the input devices that are connected to the computer (in most cases, the keyboard and a mouse). On computers that support both HP-HIL and DIN interfaces, the DIN input devices are used if both types of devices are connected.

HP-HIL input devices can plug into other HP-HIL devices, with up to seven input devices connected together. If there are no DIN input devices connected, and there are multiple HP-HIL input devices, the following algorithm is used to choose an X keyboard and pointer device.

1. If no explicit specification is made through the `X*devices` file, the last mouse (the one farthest from the computer on the HP-HIL line) is used as the X pointer and the last keyboard is used as the X keyboard.
2. If no mouse is available, the last pointing device (such as a dial box, graphics tablet, or trackball) is used as the X pointer. If no keyboard is available, the last key device (such as a buttonbox or barcode reader) is used as the X keyboard.
3. If either the pointer or keyboard are unavailable, the X server won't run unless explicitly configured to run with no input devices.

X*devices File

The X server reads an input device file, `X0devices` in `/etc/X11`, to find out what input devices it should open and attach to the display.

Note



A sample `X0devices` file is loaded into `/etc/X11` unless one already exists. In that case, it is loaded into `/usr/newconfig/etc/X11`.

The default `X0devices` file contains lines of text, but does not specify any input configuration. Rather, it assumes the default input configuration of one keyboard and one pointer.

If this is your configuration, you may not want to change the contents of the file for three reasons:

- Clients can request and receive the services of an input device regardless of whether the device is specified in a device configuration file. Thus, you need not change the `X0devices` file, or create a custom file, even though you have a custom input configuration.
- Even if you have other screen configurations, you can rely on the default input device configuration without having to create an `X*devices` file to match every `X*screens` file. For example, if you had a custom `X*screens` file, you would not necessarily need an `X*devices` file.

A custom `X*devices` file is required only when you want to tell the X server about a custom input device configuration.

5

5-4 X Windows Configuration Details

Explicitly Specifying Input Device Use

The X server can be explicitly configured to use a specific input device as the X pointer or X keyboard, or merge the data from an input device with that from the X pointer or keyboard. This configuration is done by adding information to the `X*devices` file. There is one syntax to use for HP-HIL devices, and another syntax for devices that require a device driver to be loaded by the X server (such as RS-232 devices).

HP-HIL devices can be specified in either of two ways:

- Device type and position.
- Device file name.

Explicitly Specifying RS-232 Input Device Use

Some RS-232C input devices can be used with the X server. A device driver must exist for the desired serial input device, and it must reside in the `/usr/lib/X11/extensions` directory. Input device drivers are usually supplied by the input device vendor along with the input device. Sample input device drivers and documentation describing how to write an input device driver may be found in the `/usr/contrib/X11drivers/input` directory.

To use an RS-232 input device, you must modify the `X*devices` file to inform the X server which input device driver is to be loaded, the serial port to which it is connected, and how it is to be used. This is done by adding an entry to the `X*devices` file of the following form:

```
Begin_Device_Description
Name <device_driver_name>
Path <device_file_path>
Use <device_use>
End_Device_Description
```

where:

`<device_driver_name>` Specifies the name of the input device driver shared library.

`<device_file_path>` Specifies the name of the device file for the serial port being used.

`<device_use>` Specifies the desired use of the input device, such as “keyboard”, “pointer”, “other”, or “extension”.

The following example specifies a Spatial System Spaceball[®] connected to the serial port associated with device file `/dev/tty00` as the X pointer:

```
Begin_Device_Description
Name          spaceball.sl
Path          /dev/tty00
Use           pointer
End_Device_Description
```

More examples of input device specifications for RS-232 input devices are in the `/usr/newconfig/etc/X11/X0devices` file.



5

5-6 X Windows Configuration Details

Specifying HP-HIL Input Device Use by Device Type and Position

The device can be specified using its device type and position by adding an entry to the `X*devices` file with the following form:

```
<relative_position> <device_type> <use> #<comments>
```

where:

- <relative_position>* Specifies the position of the device on the HP-HIL relative to the other devices on the HP-HIL, for example, “`first`”, “`second`”, and so on.
- <device_type>* Specifies the type of input device, such as “`keyboard`”, “`mouse`”, or “`tablet`”.
- <use>* Is “`keyboard`”, “`mouse`”, or “`other`”.
- #<comments>* Describes device. Comments are optional, but if present, must start with a “`#`”.

Separate the parts of your entry with tabs or spaces.

The position of an input device on the HP-HIL is relative to other devices of the same type. For example if you have two keyboards, a graphics tablet, and a mouse connected, they are referred to as “`first keyboard`”, “`second keyboard`”, “`first tablet`”, and “`first mouse`”.

This syntax is useful for computers on which a single X server is running, and on which no other programs directly access input devices. With this syntax, if you add a new input device to the HP-HIL, you don't have to edit the `X*devices` file unless the device is of the same type as one already named in the file and you add the device ahead of the existing device.

This syntax should not be used if more than one X server will be run on the same computer, or if non-X programs will be directly accessing input devices. The X server interprets “`first`” to mean “`first accessible`”, so you may not always get the first on the HP-HIL, just the first one not already in use.

Selecting Values for X*devices Files

X*devices files use the following special names for positions, devices, and uses:

Table 5-1. Values for X*devices Files

Positions	Device Type (Device Class)	Uses
first	keyboard (keyboard)	keyboard
second	mouse (pointer)	pointer
third	tablet (pointer)	other
fourth	buttonbox (keyboard)	
fifth	barcode (keyboard) ¹	
sixth	one_knob (pointer)	
seventh	nine_knob (pointer) ²	
	quadrature (pointer)	
	touchscreen (pointer)	
	trackball (pointer) ³	
	null	

5

1. Note also that the HP barcode reader has two modes: keyboard and ASCII. The modes are set via switches on the reader. If you set the barcode reader to ASCII transmission mode, it appears to the server as a barcode reader and the device name is therefore **barcode**. However, if you set the barcode reader to emulate a keyboard, the barcode reader appears as a keyboard and the device name should therefore be **keyboard**. What distinguishes a barcode reader set to keyboard mode from a real keyboard is the relative position or the device file name, depending on which syntax you use.
2. The nine-knob box appears to the X server as three separate input devices. Each row of knobs is a separate device, and the first device is the bottom row.
3. Similar to the barcode reader, the trackball appears to the server, not as a trackball, but as a mouse. Therefore, to specify a trackball, use the **mouse** device name. Again, what specifies the trackball instead of the real mouse is the relative position or the device filename, depending on which syntax you use.

5-8 X Windows Configuration Details

Examples

You can create a system on which the X server runs, but which does not have any input devices. In this case, clients could be run from a remote terminal, or from a remote host, and their output directed to the X server. To create a system with no input, include the following lines in the `X0devices` file:

```
first    null    keyboard
first    null    pointer
```

If you had a more complicated configuration, such as two graphics tablets, two keyboards, and a barcode reader, your `X*devices` file could look like this:

- `first tablet pointer` The pointer
- `second tablet other` Merged with the pointer
- `first keyboard other` Merged with the keyboard
- `second keyboard keyboard` The keyboard
- `first barcode other` Merged with the keyboard

In this example, the first tablet acts as the pointer, the second keyboard acts as the keyboard, input from the second tablet is treated as if it came from the X pointer, and input from the first keyboard and the barcode reader is treated as if it came from the X keyboard.

Note that the barcode reader is in ASCII mode in this example. If the barcode reader were in keyboard mode, the last line of the example would read as follows:

```
third    keyboard    other
```

More examples can be found in the `X0devices` file in `/usr/newconfig/etc/X11`.

Specifying HP-HIL Input Device Use by Device File Name

The device can be specified using the name of the device to which it is attached. This can be done by adding an entry to the `X*devices` file with the form:

```
/⟨path⟩/⟨device_file⟩ ⟨use⟩ #⟨comments⟩
```

where:

⟨path⟩/⟨device_file⟩ Specifies the name of the device file associated with the input device.

⟨use⟩ is “keyboard”, “pointer”, or “other”.

#⟨comments⟩ Describes the device. Comments are optional, but if present, must be preceded by a “#”.

This syntax should be used if more than one X server will be running on the computer, or if non-X programs will be accessing the input devices. It refers to a specific position on the HP-HIL.

Redefining the HP-HIL Search Path

The `X*devices` file can be used to redefine the path searched for HP-HIL devices. By default, the path searched is `/dev/hil`. The device files are named by appending the numbers “1” through “7” to the path.

The path is redefined by adding an entry to the `X*devices` file with the following form:

```
⟨path⟩ ⟨hil_path⟩ #⟨comment⟩
```

where:

⟨path⟩ Specifies the path to be searched for the HP-HIL input devices.

#⟨comments⟩ Describes the path. Comments are optional, but if present, must be preceded by a “#”.

The X server appends the numbers “1” through “7” to the specified path. For example, specifying:

```
    /tmp/fred    hil_path
```

results in the device names `/tmp/fred1`, `/tmp/fred2`, and so on.

5-10 X Windows Configuration Details

Stopping the X Window System

After stopping all application programs, stop the window system by holding down the **Ctrl** and left **Shift** keys, and then pressing the **Reset** key. This stops the display server, and with it the window system. (If you have a PC-style keyboard, press **Shift** **Ctrl** **Pause** instead.)

The sequence of keys that stops the display server can be customized in the `X*pointerkeys` file. Refer to the `XOpointerkeys` file in `/etc/X11`.

Initializing the Colormap with `xinitcolormap`

The `xinitcolormap` client initializes the X colormap. Specific X colormap entries (pixel values) are made to correspond to specified colors. An initialized colormap is required by applications that assume a predefined colormap (for example, many applications that use Starbase graphics).

`xinitcolormap` has the following syntax:

```
xinitcolormap [options]
```

where the *options* are:

- `-f colormapfile` Specifies a file containing a colormap.
- `-display display` Specifies the server to connect to.
- `-c count` Only the first *count* colors from the colormap file will be used if this parameter is specified.
- `-k` or `-kill` Deallocate any colormap entries that were allocated by a previous run of `xinitcolormap`.

`xinitcolormap` chooses a colormap file in the order shown below. Once one is found, then the other sources aren't searched.

1. The command line option [`-f colormapfile`].
2. `.Colormap` default value.
3. The `xcolormap` file in `/usr/lib/X11`.
4. If no colormap file is found, this default colormap specification is assumed—black (colormap entry 0), white, red yellow, green, cyan, blue, magenta (colormap entry 7).

`xinitcolormap` should be the first client program run at the start of a session in order to assure that colormap entries have the color associations specified in the colormap file. Sometimes you may encounter this X toolkit warning:

```
X Toolkit Warning: cannot allocate colormap entry for 94c4d0
```

where “94c4d0” is a color specified in the application running. If this occurs, it means that you have probably reached the limit of colors for your graphics card/display combination. Executing `xinitcolormap` may solve the problem.

For more information about `xinitcolormap`, refer to its reference page.

5-12 X Windows Configuration Details

Customizing the Mouse and Keyboard

This section describes the following customizations:

- Changing mouse button actions.
- The `xmodmap` client.
- Going mouseless.
- Customizing keyboard input.

Related information:

- Chapter 7 contains `mwm` mouse and keyboard bindings.

Changing Mouse Button Actions

Normally, the mouse pointer buttons are mapped as follows:

Table 5-2. Default Mouse Button Mapping

Button Number	Button on a 2-button mouse	Button on a 3-button Mouse
Button 1	Left button	Left button
Button 2	Both buttons simultaneously	Middle button
Button 3	Right button	Right button
Button 4		Left and middle buttons simultaneously
Button 5		Middle and right buttons simultaneously

However, you can change these mappings. To generate buttons 4 and 5 on a three-button mouse, you must enable button chording as described later in this chapter.

Table 5-3. Alternative Mouse Button Mappings

To press:	Left-Handed Mapping		OSF/Motif Mapping	
	2-button mouse	3-button mouse	2-button mouse	3-button mouse
Button 1	Right button	Right button	Left button	Left button
Button 2	Both buttons simultaneously	Middle button	Right button	Middle button
Button 3	Left button	Left button	Both buttons simultaneously	Right button
Button 4		Middle and right buttons simultaneously		Left and middle buttons simultaneously
Button 5		Middle and left buttons simultaneously		Right and middle buttons simultaneously

5

The `xmodmap` utility can be used to change mouse button mappings. The syntax for changing mouse button mappings with `xmodmap` is:

```
xmodmap {-e "pointer = {default | number [number ... ]}" | -pp}
```

`-e` Specifies a remapping expression. Valid expressions are covered in “Customizing Keyboard Input”.

`default` Set mouse keys back to default bindings.

`number` Specifies a list of button numbers to map the mouse keys to. The order of the numbers refers to the original button mapping.

`pp` Print the current pointer mapping.

For example, to reverse the positions of buttons 1 and 3 for left-handed mapping:

```
xmodmap -e "pointer = 3 2 1"      (2-button mouse)
xmodmap -e "pointer = 3 2 1 5 4"  (3-button mouse)
```

5-14 X Windows Configuration Details

To establish OSF/Motif-standard button mapping:

```
xmodmap -e "pointer = 1 3 2"      2-button mouse
xmodmap -e "pointer = 1 3 2 4 5"  3-button mouse
```

Going Mouseless with the X*pointerkeys File

Your work situation may lack sufficient desk space to adequately use a mouse pointer. You may, therefore, want to “go mouseless” by naming the keyboard (or some other input device) as the pointer.

To go mouseless, you need to have the proper configuration specified in the X*devices file and to have a special configuration file named X*pointerkeys. The default X*pointerkeys file is X0pointerkeys in /usr/lib/X11.

The X*pointerkeys file lets you specify:

- The keys that move the pointer.
- The keys that act as pointer buttons.
- The increments for movement of the pointer.
- The key sequence that resets X11.
- The pixel threshold that must be exceeded before the server switches screens.
- That button chording is enabled or disabled.
- That button latching is enabled or disabled.
- Tablet subsetting.
- Screen switching behavior for multi-screen configurations.

If you modify a X*pointerkeys file, it does not take effect until you restart the X server.

Configuring X*devices for Mouseless Operation

If you have only one keyboard and no pointer device, and you want the keyboard to serve as both keyboard and pointer, you don't have to change the default configuration of X0devices. The default input device configuration automatically assigns the pointer to the keyboard if a pointer can't be opened by the server.

If you have two or more input devices, you may need to explicitly specify which device should be the keyboard and which the pointer.

The Default Values for the X*pointerkeys File

By default, when you configure your keyboard as the pointer, the X server chooses certain number pad keys and assigns them mouse operations. Some number pad keys are assigned to pointer movement; other number pad keys are assigned to button operations.

If you don't need to change the pointer keys from their default specifications, you don't need to do anything else to use your keyboard as both keyboard and pointer. However, if you need to change the default pointer keys, you must edit the X0pointerkeys file or create a new X*pointerkeys file. The X*pointerkeys file is the file that specifies which keys are used to move the pointer when you use the keyboard as the pointer.

The default key assignments are listed in the tables in the following section on customizing the X*pointerkeys file.

Creating a Custom X*pointerkeys File

You need to modify the existing X0pointerkeys file only if one or more of the following statements are true:

- You want to use the keyboard for a pointer.
- You want to change the pointer keys from their default configuration.
- You use the X0screens file to configure your display.

You need to create a custom X*pointerkeys file only if the following statements are true:

- You want to use the keyboard for a pointer.
- You want to change the pointer keys from their default configuration.
- You use a configuration file other than the X0screens file to configure your display.

Syntax. You assign a keyboard key to a mouse function (pointer movement or button operation) by inserting a line in the X*pointerkeys file. Lines in the X*pointerkeys file have the syntax:

```
<function> <keyname> [# <comment>]
```

5-16 X Windows Configuration Details

Assigning Mouse Functions to Keyboard Keys. You can assign any mouse function, either a pointer movement or a button operation, to any keyboard key. However, make sure that the key you are assigning doesn't already serve a vital function.

You can assign keyboard keys to pointer directions by specifying options in an `X*pointerkeys` file. The following table lists the pointer movement options, the `X*pointerkeys` functions that control them, and their default values:

Table 5-4. Pointer Movement Functions

Movement Option	Function	Default Key
Move the pointer to the left.	<code>pointer_left_key</code>	<code>keypad_1</code>
Move the pointer to the right.	<code>pointer_right_key</code>	<code>keypad_3</code>
Move the pointer up.	<code>pointer_up_key</code>	<code>keypad_5</code>
Move the pointer down.	<code>pointer_down_key</code>	<code>keypad_2</code>
Add a modifier key to the pointer direction keys.	<code>pointer_key_mod1</code>	(no default)
Add a second modifier key to the pointer direction keys.	<code>pointer_key_mod2</code>	(no default)
Add a third modifier key to the pointer direction keys.	<code>pointer_key_mod3</code>	(no default)

Note that the pointer direction keys are the *keypad* number keys on the right side of the keyboard, not the *keyboard* number keys above the text character keys.

You can assign keyboard keys to pointer distances by specifying options in a `XOpointerkeys` file. The following table lists the options that determine the distance of pointer movements, the `X*pointerkeys` functions that control them, and their default value:

Table 5-5. Pointer Distance Functions

Movement	Function	Default
Move the pointer a number of pixels	<code>pointer_move</code>	10 pixels
Move the pointer using a modifier key	<code>pointer_mod1_amt</code>	40 pixels
Move the pointer using a modifier key	<code>pointer_mod2_amt</code>	1 pixel
Move the pointer using a modifier key	<code>pointer_mod3_amt</code>	5 pixels
Add a modifier to the distance keys	<code>pointer_amt_mod1</code>	no default
Add a modifier to the distance keys	<code>pointer_amt_mod2</code>	no default
Add a modifier to the distance keys	<code>pointer_amt_mod3</code>	no default

You can assign keyboard keys to mouse button operations by specifying options in a `X*pointerkeys` file. The following table lists the button operations, the `X*pointerkeys` functions that control them, and their default values:

Table 5-6. Button Operation Functions

Button Operation	Function	Default Key
Perform button 1 operations	<code>pointer_button1_key</code>	<code>keypad_*</code>
Perform button 2 operations	<code>pointer_button2_key</code>	<code>keypad_/_</code>
Perform button 3 operations	<code>pointer_button3_key</code>	<code>keypad_+</code>
Perform button 4 operations	<code>pointer_button4_key</code>	<code>keypad_-</code>
Perform button 5 operations	<code>pointer_button5_key</code>	<code>keypad_7</code>

You can change the mapping of buttons on the pointer by using options in the `X*pointerkeys` file. The following table lists the `X*pointerkeys` functions that control button mapping and their default values. Like `xmodmap` and `xset`, these functions affect only the X pointer, not any extension input devices.

Table 5-7. Button Mapping Functions

Button Mapping	Function	Default Key
Set button 1 value	<code>button_1_value</code>	1
Set button 2 value	<code>button_2_value</code>	2
Set button 3 value	<code>button_3_value</code>	3
Set button 4 value	<code>button_4_value</code>	4
Set button 5 value	<code>button_5_value</code>	5

You can change the key sequence that exits the X Window System. Also, if you use both image and overlay planes, you can change the distance you must move the pointer before you switch planes. The following table lists these options, the `X*pointerkeys` functions that control them, and their default values:

Table 5-8. Reset and Threshold Functions

Option	Function	Default Key
Exit the X Window System	<code>reset</code>	break
Add a modifier to the exit key	<code>reset_mod1</code>	control
Add a modifier to the exit key	<code>reset_mod2</code>	left_shift
Add a modifier to the exit key	<code>reset_mod3</code>	no default
Set the threshold for changing between screens	<code>screen_change_amt</code>	30 pixels (0 if a graphics tablet is used)

`screen_change_amt` is used only if your system is configured for more than one screen. `screen_change_amt` enables you to avoid switching from one screen to another if you accidentally run the pointer off the edge of the screen.

`screen_change_amt` establishes a “distance threshold” that the pointer must exceed before the server switches screens. As the previous table shows, the default width of the threshold is 30 pixels, but acceptable values range from 0 to 255.

When a graphics tablet is used as the X pointer, the `screen_change_amt` defines an area at the left and right edges of the tablet surface that will be used to control screen changes. Moving the puck or stylus into the left or right area will cause the X server to switch to the previous or next screen.

Table 5-9. Button Chording

Option	Function	Default Action
Turn button chording off or on	<code>button_chording</code>	On for devices with two buttons, off for devices with more than two buttons

Button chording refers to the generation of a button-press by pressing two other buttons. If you have a two-button mouse, you can generate Button 3 by pressing both buttons together. With a three-button mouse, you can generate button 4 by pressing the left and middle buttons together and button 5 by pressing the middle and right buttons together. See the button chording examples in the `X*pointerkeys` file.

You can also use the `X*pointerkeys` file to configure pointer buttons so they are latched. When this feature is enabled, a button you press stays logically down until you press it again. See the example `X*pointerkeys` file in `/usr/lib/X11` for information on configuring this functionality.

Note



The sample `X*pointerkeys` file is placed in `/usr/lib/X11` at install time. If you subsequently update your system, the `X*pointerkeys` file in `/usr/lib/X11` is *not* overwritten, and the sample file is placed in `/usr/newconfig`.

5-20 X Windows Configuration Details

Table 5-10. Specifying a Portion of a Tablet

Option	Function	Default
Use a subset of the tablet surface as the X pointer device	<code>tablet_subset_width</code> <code>tablet_subset_height</code> <code>tablet_subset_xorigin</code> <code>tablet_subset_yorigin</code>	disabled

If a tablet is used as the X pointer device, it may be desirable to use only a portion of the tablet surface. A rectangular subset of the surface may be specified with these functions. The units are in millimeters from the upper left corner of the tablet surface. For example, if you want to use only an “A” size portion of a larger “B” size tablet, the following lines could be added to the `X*pointerkeys` file:

```
tablet_subset_xorigin 68
tablet_subset_yorigin 40
tablet_subset_width   296
tablet_subset_height  216
```

You can also use the `X*pointerkeys` file to control screen switching behavior in multi-screen configurations. See the example `X*pointerkeys` file in `/usr/lib/X11` for an example of this functionality.

Note



The sample `X*pointerkeys` file is placed in `/usr/lib/X11` at install time. If you subsequently update your system, the `X*pointerkeys` file in `/usr/lib/X11` is *not* overwritten, and the sample file is placed in `/usr/newconfig`.

Modifier Keys. You can select up to three keys from among the two `(Shift)` keys, the two `(Extend Char)` keys, and the `(Ctrl)` key and use them each as modifier keys. A modifier key is a key that, when you hold it down and press another key, changes the meaning of that other key.

Modifier keys in the `X*pointerkeys` file have three functions:

- They specify that a certain operation can't take place until they are pressed.
- They enable you to adjust the distance covered by the pointer during a movement operation.
- They enable you to change the key sequence that exits you from X11.

For example, you can overcome the problem in the last example by assigning the left **(Shift)** key as a modifier to the pointer direction keys. Now, to move the *hpterm cursor* to the right, you press **(→)** as usual. To move the *x server pointer* to the right, you press left **(Shift)** **(→)**.

Specifying Pointer Keys. To find out what key names are valid for the keyboard you are using, enter

```
xmodmap -pk
```

You may also use the *default* X Keysymbol names assigned to these keys by the X Server.

Examples. If you only have one keyboard and no mouse, and you can live with the default pointer key assignments, you don't have to do anything else to configure your system for mouseless operation. To move the pointer to the left 10 pixels, you would press the **(1)** key on the keypad. To press mouse button 1 you would press the **(*)** key on the keypad.

However, suppose you wanted to move only one pixel to the left. Although the default value of `pointer_mod2_amt` is one pixel, no key is assigned to the modifier for that amount. Thus, you would need to edit the `XOpointerkeys` file (or create an `X*pointerkeys`) to include a line assigning one of the modifier keys to `pointer_amt_mod2`. The following line in `XOpointerkeys` assigns the left **(Shift)** key to `pointer_amt_mod2`:

```
###pointerfunction key
pointer_amt_mod2 left_shift
```

Or suppose you wanted to set up your `XOpointerkeys` file so that you could move 1, 10, 25, and 100 pixels. The following lines show one way to specify this:

```
###pointer function      key
pointer_amt_mod1         left_extend
pointer_amt_mod2         left_shift
pointer_amt_mod3         control
pointer_move              1_pixels
pointer_mod1_amt         10_pixels
pointer_mod2_amt         25_pixels
pointer_mod3_amt         100_pixels
```

With these lines in effect, one press of the **(1)** key on the keypad moves the pointer 1 pixel to the left. Pressing the left **(Extend Char)** and **(1)** moves the pointer 10 pixels

5-22 X Windows Configuration Details

to the left. Pressing left **(Shift) (↑)** moves the pointer 25 pixels to the left. And pressing **(Ctrl) (↑)** moves the pointer 100 pixels to the left.

Or, take the case, previously mentioned, where you want to use the arrow keys for both text cursor and mouse pointer. You could insert the following lines in your `X0pointerkeys` file:

```
###pointer function      key
pointer_key_mod1        left_shift
pointer_left_key        cursor_left
pointer_right_key       cursor_right
pointer_up_key          cursor_up
pointer_down_key        cursor_down
```

The above lines enable you to use the arrow keys for cursor movement, while using the shifted arrow keys for pointer movement. Note that only the left **(Shift)** key (and not the right **(Shift)**) modifies the press of an arrow key from cursor to pointer movement.

Now, suppose you want to use the arrow keys to operate the pointer, and you also need the arrow keys to control the cursor in an `hpterm` window. Furthermore, another application uses the shift-arrow key sequence to control its cursor.

The easiest way to solve this dilemma is to call in another modifier. The following lines illustrate this. Compare them to the previous example.

```
###pointer function      key
pointer_key_mod1        left_shift
pointer_key_mod2        left_extend
pointer_left_key        cursor_left
pointer_right_key       cursor_right
pointer_up_key          cursor_up
pointer_down_key        cursor_down
```

In this example,

- Pressing the **(↑)** key moves the `hpterm text cursor` up.
- Pressing left **(Shift) (↑)** moves the *cursor* up in the program you frequently operate.
- Pressing left **(Shift) left (Extend Char) (↑)** moves the *pointer* up.

Using a similar technique, you can also reassign the **(Ctrl) left (Shift) (Reset)** sequence that aborts a session. You can specify the press of a single key or a combination

of two, three, or four key presses. Just make sure that the key sequence you select isn't something you're going to type by accident.

Customizing Keyboard Input

Besides remapping the mouse's pointer and buttons to your keyboard, you can remap any key on the keyboard to any other key.

Modifying Modifier Key Bindings with `xmodmap`

To change the meaning of a particular key for a particular X11 session, or to initialize the X server with a completely different set of key mappings, use the `xmodmap` client.

Note



Note

There are now two keyboards available for Hewlett-Packard workstations, the 46021 keyboard, and the C1429 keyboard. See "Using the Keyboards" for more information on using these keyboards and the differences between them.

The syntax for `xmodmap` is as follows:

```
xmodmap <options> [<filename>]
```

where <options> are:

- display <host>:<display> Specifies the host, display number, and screen to use.
- help Displays a brief description of `xmodmap` options.
- grammar Displays a brief description of the syntax for modification expressions.
- verbose Prints log information as `xmodmap` executes.
- quiet Turns off verbose logging. This is the default.
- n Lists changes to key mappings without actually making those changes.
- e <expression> Specifies a remapping expression to be executed.
- pm, -p Prints the current modifier map to the standard output. This is the default.
- pk Prints the current keymap table to the standard output.
- pp Print the current pointer map to the standard output.
- Specifies that the standard input should be used for the input file.
- <filename> Specifies a particular key mapping file to be used.

Specifying Key Remapping Expressions

Whether you remap a single key “on the fly” with a command-line entry or install an entire new keyboard map file, you must use valid expressions in your specification, one expression for each remapping.

A valid expression is any one of the following:

Table 5-11. Valid `xmodmap` Expressions

To do this ...	Use this expression ...
Assign a key symbol to a keycode	<code>keycode <keycode> = <keysym></code>
Replace a key symbol expression with another.	<code>keysym <keysym> = <keysym></code>
Clear all keys associated with a modifier key.	<code>clear <modifier></code>
Add a key symbol to a modifier.	<code>add <modifier> = <keysym></code>
Remove a key symbol from a modifier.	<code>remove <modifier> = <keysym></code>

keycode Refers to the numerical value that uniquely identifies each key on a keyboard. Values may be in decimal, octal, or hexadecimal.

keysym Refers to the character symbol name associated with a keycode; for example, `KP_Add`.

<modifier> Specifies one of the eight modifier names: `Shift`, `Control`, `Lock`, `Mod1`, `Mod2`, `Mod3`, `Mod4`, and `Mod5`.

5

On Hewlett-Packard keyboards, the `lock` modifier is set to the `Caps Lock` key. However, any of the modifiers can be associated with any valid key symbol. Additionally, you can associate more than one key symbol with a modifier (such as `Lock = Shift_R` and `Lock = Shift_L`), and you can associate more than one modifier with a key symbol (for example, `Control = Caps_Lock` and `Lock = Caps_Lock`).

For example, on a PC-style keyboard, you can press `[D]` to print a lower case “d”, `[Shift][D]` to print a capital “D”, `[Alt][D]` to print something else, and `[Shift][Alt][D]` to print still something else.

The `xmodmap` client gives you the power to change the meaning of any key at any time or to install a whole new key map for your keyboard.

5-26 X Windows Configuration Details

Examples

Suppose you frequently press the **Caps Lock** key at the most inopportune moments. You could remove the **Caps Lock** lock key from the lock modifier, swap it for the **f1** key, then map the **f1** key to the lock modifier. Do this by creating a little swapper file that contains the following lines:

```
!This file swaps the [Caps] key with the [F1] key.  
  
remove Lock = Caps_Lock  
keysym Caps_Lock = F1  
keysym F1 = Caps_Lock  
add Lock = Caps_Lock
```

Note the use of the ! in the file to start a comment line. To put your “swapper” file into effect, enter the following on the command line:

```
xmodmap swapper
```

If you use such a swapper file, you should probably have an unswapper file. The following file enables you to swap back to the original keyboard mapping without having to exit X11:

```
!This file unswaps the [F1] key with the [Caps] key.  
  
remove Lock = Caps_Lock  
keycode 88 = F1  
keycode 55 = Caps_Lock  
add Lock = Caps_Lock
```

Note the use of the hexadecimal values to reinitialize the keycodes to the proper key symbols. You put your “unswapper” file into effect by entering the following command line:

```
xmodmap unswapper
```

On a larger scale, you can change your current keyboard to a Dvorak keyboard by creating a file with the appropriate keyboard mappings.

```
xmodmap .keymap
```

Printing a Key Map

The `-pk` option prints a list of the key mappings for the current keyboard.

```
xmodmap -pk
```

The list contains the keycode and up to four 2-part columns. The first column contains unmodified key values, the second column contains shifted key values, the third column contains meta (Extend Char) key values, and the fourth column contains shifted meta key values. Each column is in two parts: hexadecimal key symbol value, and key symbol name.

Using the Keyboards

There are now two keyboards available for Hewlett-Packard workstations. In addition to the 46021 keyboard, a personal computer-style keyboard, C1429 is also available. This new keyboard is also known as the “Enhanced Vectra” keyboard.

Understanding the Keyboards

If an application is reading input directly from the keyboard, it receives a *keycode* when a key is pressed. Equivalent keys on the two keyboards are those that generate the same keycode. If an equivalent key does not exist, there is no way to generate the corresponding keycode.

In an X Window System environment, keycodes are mapped into *key symbols* by the X library. The key symbols are stored in a *keysym table*. Application programs then reference these key symbols when accessing keys.

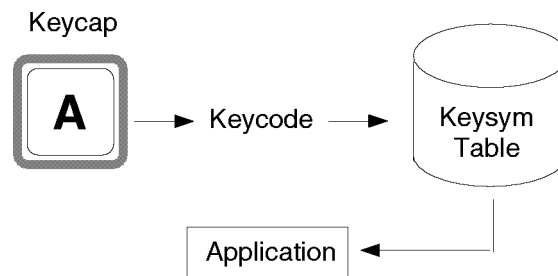


Figure 5-1. Keycap, Keycode, and Keysym Relationships

Equivalent keys are those keys that are mapped to the same key symbol. One advantage of this mapping is that if a key does not physically exist on a keyboard, its equivalent key symbol can be mapped to some other key through the corresponding keycode.

Default Keyboard Mapping

The default keyboard mapping supplied with the X Window environment maps the C1429 keyboard to the same key symbols that are used for the 46021 keyboard. This allows existing X client programs that expect to receive input from a 46021 keyboard to be used with either keyboard. However, the result is that some keys on the C1429 keyboard are mapped to key symbols that do not match the engravings on their keycaps.

Equivalent Keys

Some applications may expect to use keys that exist on one of the keyboards but not the other. In most cases, if a key does not exist on the keyboard in use, it is still possible to use some other key that is equivalent. To do this, it is necessary to know which keys are equivalent on the two keyboards.

There are 14 keys on the C1429 keyboard that generate keycodes equivalent to keys on the 46021 keyboard, but have different engravings on the keycaps. Some have the same key symbol on both keyboards, while others do not. These C1429 keys, their 46021 equivalents, and the corresponding symbol names are shown in the following table.

Table 5-12.

C1429 Keycap	46021 Keycap	Default Key Symbol	XPCmodmap Symbol
f9	blank1	F9	F9
f10	blank2	F10	F10
f11	blank3	F11	F11
f12	blank4	F12	F12
Print Screen/SysRq	Menu	Menu	Print
Scroll Lock	Stop	Cancel	Scroll_Lock
Pause/Break	Break/Reset	Break/Reset	Pause/Break
Page Up		Prior	Prior
Num Lock	System/User	System/User	Num_Lock
End	Select	Select	End
Page Down		Next	Next
Enter	Return	Return	Return
Alt (left)	Extend Char (left)	Meta_L	Alt_L
Alt (right)	Extend Char (right)	Meta_R	Alt_R

Changing Key Mapping

X provides the means to change the key mapping, if you so desire. One way to accomplish this is by running the `xmodmap` client program. Hewlett-Packard provides two files in the directory `/usr/lib/X11` to use with `xmodmap`. One, `XPCmodmap`, causes `xmodmap` to change the key mapping to match the keycap engravings on the C1429 keyboard. The other, `XHPmodmap`, causes `xmodmap` to change the key mapping to match the keycap engravings on the 46021 keyboard, which are the defaults. This allows either keyboard to be used with applications that expect the other keyboard, although only one mapping can be used at any given time. When the mapping is changed, the X Server notifies all clients that are executing at that time. Some clients may load the new mapping from the server right away, but others may have to be restarted in order to recognize the new mapping. For more information about using the `xmodmap` client, see the `xmodmap` man page.

C1429 Keyboard

Execute the following command to change the mapping of the keys shown above to match the engravings on the C1429 keycaps.

```
/usr/bin/X11/xmodmap /usr/lib/X11/XPCmodmap
```

46021 Keyboard

Execute the following command to change the mapping to match the 46021 keyboard.

```
/usr/bin/X11/xmodmap /usr/lib/X11/XHPmodmap
```

Comparing the Keyboards

The 46021 keyboard has 107 keys, while the C1429 keyboard has 101 keys. There are 7 keys on the 46021 keyboard whose keycodes cannot be generated by any key on the C1429 keyboard, and whose key symbols cannot be generated when using the default keymap for the C1429 keyboard. The missing keys are:

- **Clear Line**
- **Clear Display**
- **Insert Line**
- **Delete Line**
- **Print/Enter**
- **0** (on number pad)
- **Tab** (on number pad)

0 and **Tab** exist elsewhere on the C1429 keyboard, and the others are not needed by most applications. Applications that do need one or more of them must assign their key symbols to the keycodes of existing keys. The `xmodmap` client can be used to determine the keycode-to-key symbol mapping of existing keys, and it can also be used to assign the key symbol to the desired keycode. These keys use HP specific key symbol names whose correct spelling can be found in the file `/usr/lib/X11/XKeysymDB`.

The right **Ctrl** key on the C1429 keyboard generates a keycode that has no equivalent on the 46021 keyboard. This key has the same effect as the left **Ctrl** key by default.

Keys not mentioned above exist on both keyboards, and have the same key symbols.

PowerShade: Enhanced 3D Rendering in Software

PowerShade is a software product that allows lighting, shading, and hidden-surface removal. It offers the capability for both surface rendering and volumetric rendering. (Volumetric rendering is available on supported devices only.)

PowerShade is not supported on the GRX or on any grayscale version of the Models 705, 710, 715 or 725. (See your Owner's Guide for more details on HP 9000 workstations.)

Instructions on how to install PowerShade are included in this document. Once PowerShade has been installed, any 9.x applications that use PowerShade will run on the HP-UX 10.20 system.

Because PowerShade functionality is API-independent, it is fully supported by Starbase, HP-PHIGS, and HP PEX. For more information, refer to the appropriate API manual set.

Compatibility Considerations

You should consider the following information before you run an HP-UX 9.x application on an HP-UX 10.x system. This applies to applications that use HP PEXlib, Starbase, or HP-PHIGS graphics libraries.

- If your HP-UX 9.x application uses PowerShade functionality, you need to make sure you have PowerShade installed on your HP-UX 10.x system for it to run. To do this, use `swlist` to confirm that the PowerShade fileset has been installed on your system.
- If your HP-UX 9.x application uses HP VMX and it is designed to send output to an X terminal, you need to have PowerShade installed on your HP-UX 10.x system. Note that even though some graphics devices automatically have

PowerShade capabilities, you still need to have PowerShade installed in order to output graphics to a remote X terminal.

As part of the transition to only supporting 3D computer graphics within the X Windows environment, the Release 10.x versions of Starbase no longer support input directly from interactive devices such as a keyboard, mouse, trackball, digitizing tablet, button box, knob box, or similar devices. Applications should request input of this type through the X server.

The result of directly accessing an interactive input device from Starbase in Release 10.x is undefined. If your application reads input directly from a mouse, keyboard, etc., using Starbase input calls, you should modify that application to use Xlib calls to read data from these devices. (This statement does not apply to HP-PEXlib since that API does not include input mechanisms. The PEXlib programmer should already be using Xlib for input.)

Starbase applications that are linked with HP-UX 9.x shared or archived libraries may see different input handling behavior when run on an HP-UX 10.x system.

Versions of HP-UX prior to Release 10.x will continue to support direct input as before. Hewlett-Packard documentation will continue to describe input calls for Starbase because these manuals are also used with versions of these APIs that support direct input.

Re-Installing PowerShade

PowerShade comes bundled with the HP-UX operating system (10.30 and later), but in case you ever need to re-install it, do the following:

1. Follow the instructions in the HP-UX manual *Managing HP-UX Software with SD-UX* to install software (using the `/usr/sbin/swinstall` process).
2. Select the PowerShade product for installation.

6-2 PowerShade: Enhanced 3D Rendering in Software

HP Series 700 Graphics Performance

The following information is intended to help application developers understand graphics performance on the HP 9000 Series 700 family of graphics workstations.

The VRX family of graphics devices (for example, PersonalVRX and TurboVRX) have not changed in any significant way with the 9.0 or later releases. Therefore, the information presented here should *not* be applied to these devices. See the *HP-UX Starbase Device Drivers Manual* for more information on the VRX family.

Note



Important!

If you are not certain what devices are running on your workstation, you can obtain specific information about your graphics device (product name, device driver and capabilities supported) by executing:

```
/opt/graphics/common/bin/graphinfo
```

For a detailed description of this utility, see the manpage on `graphinfo(1G)` in the *Starbase Reference Manual*.

HP has optimized graphics performance for many typical cases. The data in the following table describes how to get the maximum performance for polygon, polyline, and polymarker primitives. In addition to primitive-specific data, there is also data regarding general rendering conditions and window configurations. It is important to note that these general rendering conditions and window configurations must be considered in addition to the primitive-specific conditions in order to make an accurate analysis of performance.

Maximum performance can be achieved using features listed under “Performance Optimized For” in the following table. Any combination of these features can be used for optimal performance. In general, performance will slowly degrade as more of these features are used. (For example, rendering with seven directional light sources is slower than with only a single one.)

Other features are available (listed as “Factors Affecting Performance”) but should be used with discretion as performance is significantly slower if even one of these features is used.

Note

The following table was complete as of the date of publication of this document. For updated table entries that provide more performance/optimization information, see the online release notes in

/opt/graphics/starbase/PERF_NOTES

Table 6-1. Optimized vs. Normal 3D Performance

Performance Optimized For	Factors Affecting Performance
Rendering Conditions	
<ul style="list-style-type: none"> ■ Up to 15 directional lights plus ambient. ■ Directional or positional eyepoint. ■ Specular reflections on or off. ■ Back face cull on or off, and with or without supplied geometric normal. ■ Clip check trivial accept or reject. ■ Perspective or parallel projections. ■ Isotropic modeling matrix (that is, angle-preserving). ■ CMAP_FULL. 	<ul style="list-style-type: none"> ■ Positional light sources. ■ Picking. ■ Heavily interleaved modal calls: <ul style="list-style-type: none"> double_buffer, vertex_format, shade_mode, hidden_surface. ■ Frequent attribute changes: <ul style="list-style-type: none"> fill_color.
Window Conditions	
<ul style="list-style-type: none"> ■ Unobscured or obscured by overlay windows only. ■ Single HP PEXlib renderer per window, or single Starbase <code>gopen</code> per window. 	<ul style="list-style-type: none"> ■ Obscured window. ■ Backing Store. ■ Multiple renderers, <code>gopens</code>. ■ Heavily interleaving Xlib calls with 3D API operations.

6

6-4 PowerShade: Enhanced 3D Rendering in Software

Table 6-1. Optimized vs. Normal 3D Performance (continued)

Performance Optimized For	Factors Affecting Performance
Polygon Primitives	
<ul style="list-style-type: none"> ■ Primitives with normals per vertex and lighting on, or primitives with RGB color data per vertex and lighting off. ■ HP PEXlib PEXFillAreaWithData, PEXSetOfFillAreaSets, PEXFillAreaSetWithData, PEXTriangleStrip. ■ Z-buffering disable. ■ Depth cue turned off. ■ Starbase polygon3d, polygon_with_data3d. ■ Starbase triangular_strip, triangular_strip_with_data, and polyhedron_with_data. ■ HP-PHIGS (one set only) fill_area_set_3_with_data. ■ HP-PHIGS triangle_strip_3_with_data ■ Normals may be unit length or not. ■ With or without move/draw edge flags, with edging off or on. ■ Convex geometry. ■ Fewer Starbase calls with more vertices per call (up to 128 vertices). ■ Geometric normals with and without vertex data. 	<ul style="list-style-type: none"> ■ Extra data actually <i>used</i> per vertex (for example, RGB and normals per vertex both used, alpha per vertex, etc.). ■ More than 64 vertices in a polygon. ■ Fill area set primitives with greater than one set. ■ Starbase partial polygons. ■ 2D polygons and fill areas or fill-area sets. ■ Device Coordinate (DC) Polygons. ■ Edged polygons.

Table 6-1. Optimized vs. Normal 3D Performance (continued)

Performance Optimized For	Factors Affecting Performance
Polyline Primitives	
<ul style="list-style-type: none"> ■ Extra data per vertex which is not used (that is, skipped). ■ Z-buffering disabled. ■ HP PEXlib, Starbase, or HP-PHIGS polyline primitives. ■ 2D or 3D transforms. ■ Depth cue turned off ■ User-supplied move/draw flags present or not present. ■ CMAP_FULL or CMAP_NORMAL (except CRX where CMAP_NORMAL is faster). ■ Fewer polylines with more vectors in each polyline. 	<ul style="list-style-type: none"> ■ Z-buffering in software using PowerShade (for example, on CRX-24, Internal Color Graphics). ■ Non-polyline vector primitives (for example, stroked text, INT_OUTLINE polygons, EDGED polygons). ■ CMAP_MONOTONIC. ■ User-supplied RGB, indirect color, or intensity per vertex. ■ Frequent line color or line type changes.
Polymarker Primitives	
<ul style="list-style-type: none"> ■ Z-buffering disabled. ■ HP PEXlib, Starbase, or HP-PHIGS polymarker primitives. ■ 2D or 3D transforms. ■ Depth cue turned off ■ User-supplied move/draw flags present or not present. ■ CMAP_FULL or CMAP_NORMAL. 	<ul style="list-style-type: none"> ■ Z-buffering in software using PowerShade (for example, on CRX-24, Internal Color Graphics). ■ CMAP_MONOTONIC (Starbase). ■ User-supplied RGB, indirect color, or intensity per vertex. ■ Pseudo color mapping method (HP-PHIGS)



6-6 PowerShade: Enhanced 3D Rendering in Software

Miscellaneous Topics

3D Thread-Safing

General Information

For HP-UX release 10.30 and later, Hewlett-Packard's 3D graphics APIs are supported in multi-threaded applications (using POSIX threads). However, these libraries are thread-restricted and can be accessed only from a single dedicated thread of a multi-threaded program. This documentation is not a tutorial on threads programming or multiprocessing application issues. For more, and general, information about the use of POSIX threads, consult the HP-UX documentation set. Further restrictions on use of these APIs in multi-threaded programs are:

- The 3D graphics libraries support kernel threads only (`libpthread`); they do not support the DCE user threads package (`libcma`).
- If your multi-threaded application uses both 3D graphics and X11, or 3D graphics and Motif routines, then the 3D graphics routine calls are restricted to the same single thread as the X11 or Motif routine calls. This restriction applies to X11 or Motif routines in any of the libraries: `libX11`, `libXext`, `libXhp11`, `libXi`, `libXt`, and `libXm`.
- Miscellaneous signal handling restrictions.

<code>SIGALRM</code>	See the “SIGALRM Details” section below for more information.
<code>SIGCHLD</code>	A multi-threaded application should not change the <code>SIGCHLD</code> action during the short periods when the graphics libraries are starting the Graphics Resource Manager daemon (“ <code>grmd</code> ”) or terminating the Starbase input daemon. See the “SIGCHLD and the GRM Daemon” and “SIGCHLD and the Starbase Input Daemon” sections below for more information.
<code>SIGPIPE</code>	A graphics application that uses an HP VISUALIZE-48/48XP device with hardware texture mapping, or an HP-PHIGS application

that does graphics input should not change the SIGPIPE signal action. See the “SIGPIPEDetails” section below for more information.

Other Threads-Related Information

1. All of the 3D graphics functions are cancellation points.
2. None of the 3D graphics functions are async-cancel safe.
3. None of the 3D graphics functions are async-signal safe.
4. None of the 3D graphics functions are fork-safe, i.e., they cannot be called by a child process after a `fork(2)`, but before an `exec(2)`.

Note



Note: Calls to 3D graphics functions between a `fork` and an `exec` have never been supported.

5. There is one situation in which graphics behavior may be different for multi-threaded versus single-threaded programs. In a multi-threaded Starbase application, a call to `gopen(3g)` a serial plotter might not return if the plotter does not respond (e.g., if the plotter is turned off). In this multi-threaded case, the graphics thread could wait forever for the device. Single-threaded behavior in this case is for the `gopen(3g)` to timeout and return an error.

SIGALRM Details

The Starbase library temporarily sets a SIGALRM signal handler and uses `setitimer(2)` to start a timer in two situations:

1. To set a timeout for device access in calls to `gopen(3g)` for a serial plotter.
2. To set a maximum time to wait for an event in calls to `await_event(3g)`, `read_choice_event(3g)`, `read_locator_event(3g)`, and `intread_locator_event(3g)`.

Calls to the above Starbase functions should not be made in one thread while at the same time another thread performs any of the following:

- Changes the SIGALRM signal action;
- Calls `sigwait(2)`, selecting the SIGALRM signal;
- Uses `setitimer(2)`;
- Uses `timer_settime(2)` to set a timer which will generate a SIGALRM signal.

7-2 Miscellaneous Topics

Possible consequences of violating these non-concurrency restrictions are:

- The Starbase function call never returns;
- The wait for a plotter response or for an event is shorter than it should be;
- Alarm signals from timers set in other threads do not have the desired effect (because the graphics signal handler is in place);
- Unpredictable results due to concurrent use of the process-wide timer provided by `setitimer(2)`.

SIGCHLD and the GRM Daemon

The Graphics Resource Manager Daemon (`grmd`) is started when the X11 Server is started. In normal operation, a Starbase, HP PEX, or HP-PHIGS application will not start the daemon, and so will not be affected by the `SIGCHLD` manipulation that occurs as part of that startup (see below). However, if the `grmd` dies for some reason, the graphics libraries will restart the daemon whenever they need shared memory. This can occur in the following instances:

- During calls to the following Starbase functions: `gopen(3g)`, `gclose(3g)`, `enable_events(3g)`, `disable_events(3g)`, `set_signals(3g)`, and `track(3g)`.
- When HP-PHIGS and HP PEX initialize their output devices.
- When HP-PHIGS input is initialized.
- During calls to `glXCreateContext` or `glXMakeCurrent`.

When the `grmd` is started, the sequence of events is:

1. Set the `SIGCHLD` action to `SIG_DFL`, saving the old action.
2. `fork(2)` and `exec(2)` an intermediate process, which is the `grmd`'s parent.
3. Call `waitpid(2)` to wait for the intermediate process to die (after starting the `grm` daemon).
4. Restore the saved `SIGCHLD` action.

Between the time that the graphics thread sets the `SIGCHLD` action to `SIG_DFL` and restores the saved action, other threads should not change the `SIGCHLD` action by calls to `sigaction(2)`, `sigvector(2)`, `signal(2)`, `sigset(2)`, or `sigwait(2)`.

The following are possible consequences of such concurrency:

- If the concurrent operation sets the SIGCHLD action to SIG_IGN, the graphics thread could hang.
- If the concurrent operation installs a signal handler for SIGCHLD, that handler may be invoked when the graphics child process dies.
- A call to `sigwait` might return in response to the death of the graphics child process.
- Any SIGCHLD action concurrently set by the application could be overwritten when the graphics thread restores the saved SIGCHLD action.

SIGCHLD and the Starbase Input Daemon

The Starbase input daemon is started whenever tracking or event monitoring is enabled. When tracking and event monitoring are turned off or when the output device is closed, Starbase terminates the daemon, using this process:

1. Set the SIGCHLD action to SIG_DFL, saving the old action.
2. Send a message to the input daemon asking it to terminate.
3. Call `waitpid(2)` to wait for the daemon's death.
4. Restore the saved SIGCHLD action.

In a Starbase application using tracking or events, a non-graphics thread should not set the SIGCHLD action by calls to `sigaction(2)`, `sigvector(2)`, `signal(2)`, `sigset(2)`, or `sigwait(2)` concurrently with calls in the graphics thread to `track(3g)`, `track_off(3g)`, `disable_events(3g)`, or `gclose(3g)`.

Possible consequences of violating this restriction are the same as those listed above for the `grmd` daemon.

SIGPIPE Details

The graphics libraries start a daemon process and communicate with that process via sockets in two situations:

- For hardware texture mapping on an HP VISUALIZE-48/-48XP display using the Texture Interrupt Manager Daemon (`timd`).
- For HP-PHIGS input using the PHIGS daemon (`phg_daemon`).

7-4 Miscellaneous Topics

When starting either of these daemons, the graphics library permanently sets the SIGPIPE action to SIG_IGN. This prevents the terminating SIGPIPE signal from being delivered to the process should the daemon die abnormally.

If your application changes the SIGPIPE action to SIG_DFL or to a specific handler, an abnormal death of either `timd` or `phg_daemon` will result in a SIGPIPE signal being delivered to the process when the graphics library next attempts to communicate with the daemon. If the action is SIG_DFL, the process will terminate.

HP CDE and HP VUE

Hewlett-Packard is in the process of a transition to a standard user environment. Two user environments were shipped with HP-UX 10.20: HP VUE and HP CDE (Common Desktop Environment). Starting with HP-UX 10.20, HP CDE was the default user environment, and although HP VUE was still available with HP-UX 10.20, it is not available in HP-UX 10.20 releases. See the *Common Desktop Environment User's Guide* for more information on HP CDE.

From a 3D graphics point of view, the change in user environments should have no effect.

Shared Memory Usage

Graphics processes use shared memory to access data pertaining to the display device and the X11 resources created by the server (for example, color maps, cursors, etc.). The X11 server initiates an independent process called the Graphics Resource Manager (GRM) to manage these resources among graphics processes. One problem encountered with GRM shared memory is that it may not be large enough to run some applications.

HP PEX, Starbase, and HP-PHIGS use GRM shared memory for VM double-buffering. If your application is running on a low-end graphics system (for example, an HP 710 or 712), you set the environment variable `HP_VM_DOUBLE_BUFFER` (or `SB_710_VM_DB`), and you have several large double-buffered windows open simultaneously, then your application could use up available GRM shared memory. If you encounter a `dbuffer_switch` error message while using VM double-buffering, you may have encountered this problem.

You can prevent this problem by changing with Shared Memory size through HP-UX's SAM (System Administration Manager) program.

Reference Documentation

You may find the following documentation helpful when using HP graphics products:

- For Starbase programming
 - *Starbase Reference*
 - *Starbase Graphics Techniques*
 - *HP-UX Starbase Device Drivers Manual*
 - *Starbase Technical Addendum for HP-UX 10.20*
 - *Starbase Display List Programmer's Manual*
 - *Fast Alpha/Font Manager Programmer's Manual*
- For PEXlib programming
 - *PEXlib Programming Manual*
 - *PEXlib Reference Manual*
 - *HP PEX Implementation and Programming Supplement*
- For HP-PHIGS programming
 - *HP-PHIGS C and Fortran Binding Reference*
 - *HP-PHIGS Graphics Techniques*
 - *HP-PHIGS Workstation Characteristics and Implementation*
 - *HP-PHIGS Technical Addendum for HP-UX 10.20*
- For installing products
 - *HP-UX Reference*
 - *System Administration Tasks*
 - *Installing and Updating HP-UX*

Reference

X Windows

A portable, network-transparent window system.

Synopsis

The X Window System is a network-transparent window system developed at MIT which runs on a wide range of computing and graphics machines. It should be relatively straightforward to build the MIT software distribution on most ANSI C-compliant and POSIX-compliant systems. Commercial implementations are also available for a wide range of platforms.

The X Consortium requests that the following names be used when referring to this software:

- X
- X Window System
- X Version 11
- X Window System, Version 11
- X11

X Window System is a trademark of the Massachusetts Institute of Technology.

X Windows

Description

X Window System servers run on computers with bit-mapped displays. The server distributes user input to and accepts output requests from various client programs through a variety of different interprocess communication channels. Although the most common case is for the client programs to be running on the same machine as the server, clients can be run transparently from other machines (including machines with different architectures and operating systems) as well.

X supports overlapping hierarchical subwindows and text and graphics operations, on both monochrome and color displays. For a full explanation of the functions that are available, refer to:

- *Xlib - C Language X Interface*,
- The *X Window System Protocol* specification,
- *X Toolkit Intrinsics - C Language Interface*, and
- The various Toolkit documents.

The number of programs that use X is quite large. Programs provided in the core MIT distribution include: a terminal emulator (**xterm**), a window manager (**twm**), a display manager (**xdm**), a console redirect program (**xconsole**), mail managing utilities (**xmh** and **xbiff**), a manual page browser (**xman**), a bitmap editor (**bitmap**), a resource editor (**editres**), a ditroff previewer (**xditview**), access control programs (**xauth** and **xhost**), user preference setting programs (**xrdb**, **xcmsdb**, **xset**, **xsetroot**, **xstdcmap**, and **xmodmap**), a load monitor (**xload**), clocks (**xclock** and **oclock**), a font displayer (**xfd**), utilities for listing information about fonts, windows, and displays (**xlsfonts**, **xfontsel**, **xwininfo**, **xlsclients**, **xdpyinfo**, and **xprop**), a diagnostic for seeing what events are generated and when (**xev**), screen image manipulation utilities (**xwd**, **xwud**, **xpr**, and **xmag**), and various demos (**xeyes**, **ico**, **xgc**, **x11perf**, etc.).

Hewlett-Packard provides a graphical user environment called The **Common Desktop Environment** (CDE). HP CDE is the user interface, enabling the user to control a workstation by directly manipulating graphic objects instead of typing commands on a command-line prompt. See the *CDE User's Guide* for complete information on HP CDE.

A

A-2 Reference

X Windows

Hewlett-Packard does not provide or support the entire core MIT distribution. Many of these programs or clients are sample implementations, or perform tasks that are accomplished by other clients in Hewlett-Packard's Common Desktop Environment. The primary differences between the core MIT distribution and the Hewlett-Packard X11 release are listed below:

- Terminal Emulation** Although `hpterm` is the primary terminal emulator, `xterm` is also provided and supported.
- Window Management** `twm` is replaced by `mwm` and `dtwm`.
- Display Manager** `xdm` is replaced by an enhanced version called `dtlogin`.
- Bitmap Editing** `bitmap` is replaced by `dticon`.
- Font Display** This is handled by the terminal emulation option "`-fn override`". `xfd` is supplied but not supported.
- Demos** Obtained from the InterWorks users group.

A number of unsupported core MIT clients and miscellaneous utilities are provided in `/usr/contrib/bin`. In addition, the entire core MIT distribution, compiled for Hewlett-Packard platforms, can be obtained from HP's users group InterWorks for a nominal fee.

Many other utilities, window managers, games, toolkits, etc. are included as user-contributed software in the MIT distribution, or are available using anonymous ftp on the Internet. See your site administrator for details.

Starting Up

Normally, the X Window System is started on Hewlett-Packard systems by `dtlogin`, which is an enhanced version of the MIT client `xdm`. `dtlogin` can be used to bring up a full CDE session, a light CDE session, or a fail-safe session that uses no other part of CDE. If `dtlogin` is not used, `xinit` may be used with `x11start`. See the reference pages for these functions for more information.

A

X Windows

Display Names

From the user's perspective, every X server has a display name of the form:

hostname : displaynumber . screennumber

This information is used by the application to determine how it should connect to the server and which screen it should use by default (on displays with multiple monitors):

hostname The hostname specifies the name of the machine to which the display is physically connected. If the hostname is not given, the most efficient way of communicating to a server on the same machine will be used.

displaynumber The phrase "display" is usually used to refer to the collection of monitors that share a common keyboard and pointer (mouse, tablet, etc.). Most workstations tend to only have one keyboard, and therefore, only one display. Larger, multi-user systems, however, will frequently have several displays so that more than one person can be doing graphics work at once. To avoid confusion, each display on a machine is assigned a display number (beginning at 0) when the X server for that display is started. The display number must always be given in a display name.

screennumber Some displays share a single keyboard and pointer among two or more monitors. Since each monitor has its own set of windows, each screen is assigned a screen number (beginning at 0) when the X server for that display is started. If the screen number is not given, then screen 0 will be used.

On POSIX systems, the default display name is stored in your `DISPLAY` environment variable. This variable is set automatically by the `xterm` terminal emulator. However, when you log into another machine on a network, you'll need to set `DISPLAY` by hand to point to your display. For example,

```
% setenv DISPLAY myws:0            (C Shell)
$ DISPLAY=myws:0; export DISPLAY (Korn Shell)
```

The `xon` script can be used to start an X program on a remote machine; it automatically sets the `DISPLAY` variable correctly.

A-4 Reference

X Windows

Finally, most X programs accept a command line option of “-display *display-name*” to temporarily override the contents of DISPLAY. This is most commonly used to pop windows on another person’s screen or as part of a “remote shell” command to start an `xterm` pointing back to your display. For example,

```
$ xload -display joesws:0 -geometry 100x100+0+0
$ rsh big xterm -display myws:0 -ls </dev/null
```

X servers listen for connections on a variety of different communications channels (network byte streams, shared memory, etc.). Since there can be more than one way of contacting a given server, the *hostname* part of the display name is used to determine the type of channel (also called a **transport layer**) to be used. X servers generally support the following types of connections:

- local The hostname part of the display name should be the empty string. For example: “:0”, “:1”, or “:0.1”. The most efficient local transport is chosen.
- TCP/IP The hostname part of the display name should be the server machine’s IP address name. Full Internet names, abbreviated names, and IP addresses are all allowed. For example: `expo.lcs.mit.edu:0`, `expo:0`, `18.30.0.212:0`, `bigmachine:1`, and `hydra:0.1`.

Access Control

An X server can use several types of access control. Mechanisms provided in Release 5 are:

- Host Access (simple host-based access control);
- MIT-MAGIC-COOKIE-1 (shared plain-text “cookies”);
- XDM-AUTHORIZATION-1 (secure DES based private-keys); and
- SUN-DES-1 (based on Sun’s secure `rpc` system).

`dtlogin/xdm` initializes access control for the server, and also places authorization information in a file accessible to the user. Normally, the list of hosts from which connections are always accepted should be empty, so that only clients with are explicitly authorized can connect to the display. When you add entries to the host list (with `xhost`), the server no longer performs any authorization on connections from those machines. Be careful with this.

The file from which Xlib extracts authorization data can be specified with the environment variable `XAUTHORITY`, and defaults to the file `.Xauthority` in the

Reference A-5

A

X Windows

home directory. `dtlogin/xdm` uses `$HOME/.Xauthority` and will create it or merge in authorization records if it already exists when a user logs in.

If you use several machines, and share a common home directory across all of the machines by means of a network file system, then you never really have to worry about authorization files; the system should work correctly by default. Otherwise, as the authorization files are machine-independent, you can simply copy the files to share them. To manage authorization files, use `xauth`. This program allows you to extract records and insert them into other files. Using this, you can send authorization to remote machines when you log in, if the remote machine does not share a common home directory with your local machine. Note that authorization information transmitted “in the clear” through a network file system or using `ftp` or `rcp` can be “stolen” by a network eavesdropper, and as such may enable unauthorized access. In many environments this level of security is not a concern, but if it is, you should know the exact semantics of the particular authorization data to know if this is actually a problem.

Geometry Specifications

One of the advantages of using window systems instead of hardwired terminals is that applications don't have to be restricted to a particular size or location on the screen. Although the layout of windows on a display is controlled by the window manager that the user is running (described below), most X programs accept a command line argument of the form:

```
-geometry widthxheight+xoff+yoff
```

(where *width*, *height*, *xoff*, and *yoff* are numbers) for specifying a preferred size and location for this application's main window.



A

A-6 Reference

X Windows

The *width* and *height* parts of the geometry specification are usually measured in either pixels or characters, depending on the application. The *xoff* and *yoff* parts are measured in pixels and are used to specify the distance of the window from the left (or right) and top (or bottom) edges of the screen, respectively. Both types of offsets are measured from the indicated edge of the screen to the corresponding edge of the window. The X offset may be specified in the following ways:

- +xoff* The left edge of the window is to be placed *xoff* pixels in from the left edge of the screen (i.e., the X coordinate of the window's origin will be *xoff*). *xoff* may be negative, in which case the window's left edge will be off the screen.
- xoff* The right edge of the window is to be placed *xoff* pixels in from the right edge of the screen. *xoff* may be negative, in which case the window's right edge will be off the screen.

The Y offset has similar meanings:

- +yoff* The top edge of the window is to be *yoff* pixels below the top edge of the screen (i.e. the Y coordinate of the window's origin will be *yoff*). *yoff* may be negative, in which case the window's top edge will be off the screen.
- yoff* The bottom edge of the window is to be *yoff* pixels above the bottom edge of the screen. *yoff* may be negative, in which case the window's bottom edge will be off the screen.

Offsets must be given as pairs; in other words, in order to specify either *xoff* or *yoff* both must be present. Windows can be placed in the four corners of the screen using the following specifications:

- *+0+0* (the upper left-hand corner)
- *-0+0* (the upper right-hand corner)
- *-0-0* (the lower right-hand corner)
- *+0-0* (the lower left-hand corner)

A



Reference A-7

X Windows

In the following examples, a terminal emulator will be placed in roughly the center of the screen and a load average monitor, mailbox, and clock will be placed in the upper right hand corner:

```
xterm -fn 6x10 -geometry 80x24+30+200 &  
xclock -geometry 48x48-0+0 &  
xload -geometry 48x48-96+0 &  
xbiff -geometry 48x48-48+0 &
```

Window Managers

The layout of windows on the screen is controlled by special programs called **window managers**. Although many window managers will honor geometry specifications as given, others may choose to ignore them (requiring the user to explicitly draw the window's region on the screen with the pointer, for example).

Since window managers are regular (albeit complex) client programs, a variety of different user interfaces can be built. The Hewlett-Packard distribution comes with window managers named **mwm** and **dtwm**, which support overlapping windows, popup menus, point-and-click or click-to-type input models, title bars, nice icons (and an icon manager for those who don't like separate icon windows).

See the user-contributed software in the MIT distribution for other popular window managers.

Font Names

Collections of characters for displaying text and symbols in X are known as **fonts**. A font typically contains images that share a common appearance and look nice together (for example, a single size, boldness, slantedness, and character set). Similarly, collections of fonts that are based on a common type face—the variations are usually called **roman**, **bold**, **italic** (or **oblique**), and **bold italic** (or **bold oblique**)—are called **families**.

Fonts come in various sizes. The X server supports scalable fonts, meaning it is possible to create a font of arbitrary size from a single source for the font. The server supports scaling from outline fonts and bitmap fonts. Scaling from outline fonts usually produces significantly better results on large point sizes than scaling from bitmap fonts.

A

A-8 Reference

X Windows

An X server can obtain fonts from individual files stored in directories in the file system, or from one or more font servers, or from a mixture of directories and font servers. The list of places the server looks when trying to find a font is controlled by its **font path**. Although most installations will choose to have the server start up with all of the commonly used font directories in the font path, the font path can be changed at any time with the **xset** program. However, it is important to remember that the directory names are on the server's machine, not on the application's. Usually, fonts used by X servers and font servers can be found in subdirectories under **/usr/lib/X11/fonts**:

- **/usr/lib/X11/fonts/iso_8859.1/75dpi**
This directory contains bitmap fonts contributed by Adobe Systems, Inc., Digital Equipment Corporation, Bitstream, Inc., Bigelow and Holmes, and Sun Microsystems, Inc. for 75 dot-per-inch displays. An integrated selection of sizes, styles, and weights are provided for each family.
- **/usr/lib/X11/fonts/iso_8859.1/100dpi**
This directory contains 100 dot-per-inch versions of some of the fonts in the 75dpi directory.

Bitmap font files are usually created by compiling a textual font description into binary form, using **bdftopcf**. Font databases are created by running the **mkfontdir** program in the directory containing the source or compiled versions of the fonts. Whenever fonts are added to a directory, **mkfontdir** should be rerun so that the server can find the new fonts. To make the server reread the font database, reset the font path with the **xset** program. For example, to add a font to a private directory, the following commands could be used:

```
$ cp newfont.pcf ~/myfonts
$ mkfontdir ~/myfonts
$ xset fp rehash
```

The **xlsfonts** program can be used to list the fonts available on a server. Font names tend to be fairly long, as they contain all of the information needed to uniquely identify individual fonts. However, the X server supports wildcarding of font names, so the full specification

“-adobe-courier-medium-r-normal--10-100-75-75-m-60-iso8859-1”

might be abbreviated as

“-*-courier-medium-r-normal--*-100-***-iso8859-1”.

A

X Windows

Because the shell also has special meanings for “*” and “?”, wildcarded font names should be quoted, as in:

```
$ xlsfonts -fn '*-courier-medium-r-normal--*-100-*-*-*-*-*'
```

The `xlsfonts` program can be used to list all of the fonts that match a given pattern. With no arguments, it lists all available fonts. This will usually list the same font at many different sizes. To see just the base scalable font names, try using one of the following patterns:

```
-----*-0-0-0-0*-0-***  
-----*-0-0-75-75*-0-***  
-----*-0-0-100-100*-0-***
```

To convert one of the resulting names into a font at a specific size, replace one of the first two zeros with a nonzero value. The field containing the first zero is for the pixel size; replace it with a specific height in pixels to name a font at that size. Alternatively, the field containing the second zero is for the point size; replace it with a specific size in decipoints (there are 722.7 decipoints to the inch) to name a font at that size. The last zero is an average width field, measured in tenths of pixels; some servers will anamorphically scale if this value is specified.

Font Server Names

One of the following forms can be used to name a font server that accepts TCP connections:

```
tcp/hostname:port  
tcp/hostname:port/cataloguelist
```

The *hostname* specifies the name (or decimal numeric address) of the machine on which the font server is running. The *port* is the decimal TCP port on which the font server is listening for connections. The *cataloguelist* specifies a list of catalogue names, with “+” as a separator. For example:

```
tcp/expo.lcs.mit.edu:7000  
tcp/18.30.0.212:7001/all.
```

A

A-10 Reference

Color Names

Most applications provide ways of tailoring (usually through resources or command-line arguments) the colors of various elements in the text and graphics they display. A color can be specified either by an abstract color name, or by a numerical color specification. The numerical specification can identify a color in either device-dependent (RGB) or device-independent terms. Color strings are case-insensitive.

X supports the use of abstract color names, for example, “red”, “blue”. A value for this abstract name is obtained by searching one or more color-name databases. Xlib first searches zero or more client-side databases; the number, location, and content of these databases is implementation-dependent. If the name is not found, the color is looked up in the X server’s database. The text form of this database is commonly stored in the file `/usr/lib/X11/rgb.txt`.

A numerical color specification consists of a color space name and a set of values in the following syntax:

color_space_name:value/.../value

An RGB Device specification is identified by the prefix “rgb:” and has the following syntax:

rgb:red/green/blue

where *red*, *green*, and *blue* are encoded as *h*, *hh*, *hhh*, or *hhhh*, and *h* represents a single hexadecimal digit.

Note that *h* indicates the value scaled in 4 bits; *hh*, the value scaled in 8 bits; *hhh*, the value scaled in 12 bits; and *hhhh* the value scaled in 16 bits, respectively. These values are passed directly to the X server, and are assumed to be gamma corrected.

X Windows

The eight primary colors can be represented as:

- Black: `rgb:0/0/0`
- Red: `rgb:ffff/0/0`
- Green: `rgb:0/ffff/0`
- Blue: `rgb:0/0/ffff`
- Yellow: `rgb:ffff/ffff/0`
- Magenta: `rgb:ffff/0/ffff`
- Cyan: `rgb:0/ffff/ffff`
- White: `rgb:ffff/ffff/ffff`

For backward compatibility, an older syntax for RGB device is supported, but its continued use is not encouraged. The syntax is an initial “pound-sign” character, followed by a numeric specification, in one of the following formats:

- `#rgb` (4 bits each)
- `#rrggbb` (8 bits each)
- `#rrrgggbbb` (12 bits each)
- `#rrrrggggbbbb` (16 bits each)

The *r*, *g*, and *b* represent single hexadecimal digits. When fewer than 16 bits each are specified, they represent the most-significant bits of the value (unlike the “`rgb:`” syntax, in which values are scaled). For example, `#3a7` is the same as `#3000a0007000`.

An RGB intensity specification is identified by the prefix “`rgbi:`” and has the following syntax:

`rgbi:red/green/blue`

The *red*, *green*, and *blue* are floating-point values between 0.0 and 1.0, inclusive. They represent linear intensity values, with 1.0 indicating full intensity, 0.5 indicating half intensity, and so on. These values will be gamma-corrected by Xlib before being sent to the X server. The input format for these values is an optional sign, a string of numbers possibly containing a decimal point, and an optional exponent field containing an “E” or “e” followed by a possibly signed integer string.

A

A-12 Reference

X Windows

The standard device-independent string specifications have the following syntax:

```
CIEXYZ: X/Y/Z (none, 1, none)
CIEuvY: u/v/Y (≈.6, ≈.6, 1)
CIExyY: x/y/Y (≈.75, ≈.85, 1)
CIELab: L/a/b (100, none, none)
CIELuv: L/u/v (100, none, none)
TekHVC: H/V/C (360, 100, 100)
```

All of the values (*C, H, V, X, Y, Z, a, b, u, v, y, x*) are floating-point values. Some of the values are constrained to be between zero and some upper bound; the upper bounds are given in parentheses above. The syntax for these values is an optional “+” or “-” sign, a string of digits possibly containing a decimal point, and an optional exponent field consisting of an “E” or “e” followed by an optional “+” or “-” sign, followed by a string of digits.

For more information on device independent color, see the Xlib reference manual.

A



Reference A-13

X Windows

Keyboards

The X keyboard model is broken into two layers: server-specific codes (called **keycodes**) which represent the physical keys, and server-independent symbols (called **keysyms**) which represent the letters or words that appear on the keys. Two tables are kept in the server for converting keycodes to keysyms:

Modifier List Some keys (such as Shift, Control, and Caps Lock) are known as **modifiers** and are used to select different symbols that are attached to a single key (such as Shift-a, which generates a capital “A”, and Control-l, which generates a control character “^L”). The server keeps a list of keycodes corresponding to the various modifier keys. Whenever a key is pressed or released, the server generates an event that contains the keycode of the indicated key as well as a mask that specifies which of the modifier keys are currently pressed. Most servers set up this list to initially contain the various shift, control, and shift-lock keys on the keyboard.

Keymap Table Applications translate event keycodes and modifier masks into keysyms using a keysym table which contains one row for each keycode and one column for various modifier states. This table is initialized by the server to correspond to normal typewriter conventions. The exact semantics of how the table is interpreted to produce keysyms depends on the particular program, libraries, and language input method used, but the following conventions for the first four keysyms in each row are generally adhered to.

The first four elements of the list are split into two groups of keysyms. Group 1 contains the first and second keysyms; Group 2 contains the third and fourth keysyms. Within each group, if the first element is alphabetic and the the second element is the special keysym **NoSymbol**, then the group is treated as equivalent to a group in which the first element is the lowercase letter and the second element is the uppercase letter.

A

Switching between groups is controlled by the keysym named “Mode Switch”, by attaching that keysym to some key and attaching that key to any one of the

A-14 Reference

X Windows

modifiers Mod1 through Mod5. This modifier is called the **group modifier**. Group 1 is used when the group modifier is off, and Group 2 is used when the group modifier is on.

Within a group, the modifier state determines which keysym to use. The first keysym is used when the Shift and Lock modifiers are off. The second keysym is used when the Shift modifier is on, when the Lock modifier is on and the second keysym is uppercase alphabetic, or when the Lock modifier is on and is interpreted as ShiftLock. Otherwise, when the Lock modifier is on and is interpreted as CapsLock, the state of the Shift modifier is applied first to select a keysym; but if that keysym is lowercase alphabetic, then the corresponding uppercase keysym is used instead.

Options

Most X programs attempt to use the same names for command line options and arguments. All applications written with the X Toolkit Intrinsics automatically accept the following options:

- display *display* This option specifies the name of the X server to use.
- geometry *geometry* This option specifies the initial size and location of the window.
- bg *color*,
-background *color* Either option specifies the color to use for the window background.
- bd *color*,
-bordercolor *color* Either option specifies the color to use for the window border.
- bw *number*,
-borderwidth *number* Either option specifies the width in pixels of the window border.
- fg *color*,
-foreground *color* Either option specifies the color to use for text or graphics.
- fn *font*, -font *font* Either option specifies the font to use for displaying text.
- iconic This option indicates that the user would prefer that the application's windows initially not be visible as if the windows had be immediately iconified by the user. Window managers may choose not to honor the application's request.

A

X Windows

- name** This option specifies the name under which resources for the application should be found. This option is useful in shell aliases to distinguish between invocations of an application, without resorting to creating links to alter the executable file name.
- rv, -reverse** Either option indicates that the program should simulate reverse video if possible, often by swapping the foreground and background colors. Not all programs honor this or implement it correctly. It is usually only used on monochrome displays.
- +rv** This option indicates that the program should not simulate reverse video. This is used to override any defaults since reverse video doesn't always work properly.
- selectionTimeout** This option specifies the timeout in milliseconds within which two communicating applications must respond to one another for a selection request.
- synchronous** This option indicates that requests to the X server should be sent synchronously, instead of asynchronously. Since Xlib normally buffers requests to the server, errors do not necessarily get reported immediately after they occur. This option turns off the buffering so that the application can be debugged. It should never be used with a working program.
- title *string*** This option specifies the title to be used for this window. This information is sometimes used by a window manager to provide some sort of header identifying the window.
- xnllanguage** This option specifies the language, territory, and codeset for use in resolving resource and other filenames.
language[_territory]
[.codeset]
- xrm *resourcestring*** This option specifies a resource name and value to override any defaults. It is also very useful for setting resources that don't have explicit command line arguments.

A

A-16 Reference

Resources

To make the tailoring of applications to personal preferences easier, X provides a mechanism for storing default values for program resources (e.g., background color, window title, etc.). Resources are specified as strings that are read in from various places when an application is run. Program components are named in a hierarchical fashion, with each node in the hierarchy identified by a class and an instance name. At the top level is the class and instance name of the application itself. By convention, the class name of the application is the same as the program name, but with the first letter capitalized, although some programs that begin with the letter “x” also capitalize the second letter for historical reasons.

The precise syntax for resources is:

```
ResourceLine = Comment | IncludeFile | ResourceSpec | empty_line
Comment      = "!" {any_character_except_null_or_newline}
IncludeFile  = "#" WhiteSpace "include" WhiteSpace FileName WhiteSpace
FileName     = valid filename for operating system
ResourceSpec = WhiteSpace ResourceName WhiteSpace ":" WhiteSpace Value
ResourceName = [Binding] {Component Binding} ComponentName
Binding      = "." | "*"
WhiteSpace   = {space | horizontal tab}
Component    = "?" | ComponentName
ComponentName = NameChar {NameChar}
NameChar     = "a"-"z" | "A"-"Z" | "0"-"9" | "_" | "-"
Value        = {any character except null or unescaped newline}
```

Elements separated by vertical bar (“|”) are alternatives. Braces (“{” ... “}”) indicate zero or more repetitions of the enclosed elements. Brackets (“[” ... “]”) indicate that the enclosed element is optional. Quotes (“ ”) are used around literal characters.

IncludeFile lines are interpreted by replacing the line with the contents of the specified file. The word “include” must be in lowercase. The filename is interpreted relative to the directory of the file in which the line occurs (for example, if the filename contains no directory or contains a relative directory specification).

If a ResourceName contains a contiguous sequence of two or more Binding characters, the sequence will be replaced with single “.” character if the sequence contains only “.” characters, otherwise the sequence will be replaced with a single “*” character.

A

X Windows

A resource database never contains more than one entry for a given ResourceName. If a resource file contains multiple lines with the same ResourceName, the last line in the file is used.

Any whitespace character before or after the name or colon in a ResourceSpec are ignored. To allow a Value to begin with whitespace, the two-character sequence “*space*” (backslash followed by space) is recognized and replaced by a space character, and the two-character sequence “*tab*” (backslash followed by horizontal tab) is recognized and replaced by a horizontal tab character. To allow a Value to contain embedded newline characters, the two-character sequence “*n*” is recognized and replaced by a newline character. To allow a Value to be broken across multiple lines in a text file, the two-character sequence “*newline*” (backslash followed by newline) is recognized and removed from the value. To allow a Value to contain arbitrary character codes, the four-character sequence “*nnn*”, where each *n* is a digit character in the range of 0-7, is recognized and replaced with a single byte that contains the octal value specified by the sequence. Finally, the two-character sequence “\” is recognized and replaced with a single backslash.

A

A-18 Reference

X Windows

When an application looks for the value of a resource, it specifies a complete path in the hierarchy, with both class and instance names. However, resource values are usually given with only partially specified names and classes, using pattern matching constructs. An asterisk (“*”) is a loose binding and is used to represent any number of intervening components, including none. A period (“.”) is a tight binding and is used to separate immediately adjacent components. A question mark (“?”) is used to match any single component name or class. A database entry cannot end in a loose binding; the final component (which cannot be “?”) must be specified. The lookup algorithm searches the resource database for the entry that most closely matches (is most specific for) the full name and class being queried. When more than one database entry matches the full name and class, precedence rules are used to select just one. The full name and class are scanned from left to right (from highest level in the hierarchy to lowest), one component at a time. At each level, the corresponding component and/or binding of each matching entry is determined, and these matching components and bindings are compared according to precedence rules. Each of the rules is applied at each level, before moving to the next level, until a rule selects a single entry over all others. The rules (in order of precedence) are:

1. An entry that contains a matching component (whether name, class, or “?”) takes precedence over entries that elide the level (that is, entries that match the level in a loose binding).
2. An entry with a matching name takes precedence over both entries with a matching class and entries that match using “?”. An entry with a matching class takes precedence over entries that match using “?”.
3. An entry preceded by a tight binding takes precedence over entries preceded by a loose binding.

A

X Windows

Programs based on the X Toolkit Intrinsic obtain resources from the following sources (other programs usually support some subset of these sources):

- **RESOURCE_MANAGER** root window property
Any global resources that should be available to clients on all machines should be stored in the **RESOURCE_MANAGER** property on the root window of the first screen using the `xrdb` program. This is frequently taken care of when the user starts X through the display manager.
- **SCREEN_RESOURCES** root window property
Any resources specific to a given screen (e.g. colors) that should be available to clients on all machines should be stored in the **SCREEN_RESOURCES** property on the root window of that screen. The `xrdb` program will sort resources automatically and place them in **RESOURCE_MANAGER** or **SCREEN_RESOURCES**, as appropriate.
- Application-specific files
Directories named by the environment variable **XUSERFILESEARCHPATH** or the environment variable **XAPPLRESDIR**, plus directories in a standard place (usually under `/usr/lib/X11`, but this can be overridden with the **XFILESEARCHPATH** environment variable) are searched for application-specific resources. For example, application default resources are usually kept in `/usr/lib/X11/app-defaults`. See the *X Toolkit Intrinsic - C Language Interface* manual for details.
- **XENVIRONMENT**
Any user- and machine-specific resources may be specified by setting the **XENVIRONMENT** environment variable to the name of a resource file to be loaded by all applications. If this variable is not defined, a file named `“$HOME/.Xdefaults-hostname”` is looked for instead, where *hostname* is the name of the host where the application is executing.
- **-xrm *resourcestring***
Resources can also be specified from the command line. The *resourcestring* is a single resource name and value as shown above. Note that if the string contains characters interpreted by the shell (e.g., asterisk), they must be quoted. Any number of **-xrm** arguments may be given on the command line.

A

A-20 Reference

X Windows

Program resources are organized into groups called **classes**, so that collections of individual resources (each of which are called **instances**) can be set all at once. By convention, the instance name of a resource begins with a lowercase letter and class name with an uppercase letter. Multiple word resources are concatenated with the first letter of the succeeding words capitalized. Applications written with the X Toolkit Intrinsics will have at least the following resources:

- **background** (class **Background**)
This resource specifies the color to use for the window background.
- **borderWidth** (class **BorderWidth**)
This resource specifies the width in pixels of the window border.
- **borderColor** (class **BorderColor**)
This resource specifies the color to use for the window border.

Most applications using the X Toolkit Intrinsics also have the resource **foreground** (class **Foreground**), specifying the color to use for text and graphics within the window.

X Windows

By combining class and instance specifications, application preferences can be set quickly and easily. Users of color displays will frequently want to set **Background** and **Foreground** classes to particular defaults. Specific color instances such as text cursors can then be overridden without having to define all of the related resources. For example,

```
dticon*Dashed:           off
XTerm*cursorColor:      gold
XTerm*multiScroll:      on
XTerm*jumpScroll:      on
XTerm*reverseWrap:     on
XTerm*curses:          on
XTerm*Font:            6x10
XTerm*scrollBar:       on
XTerm*scrollbar*thickness: 5
XTerm*multiClickTime:  500
XTerm*charClass:       33:48,37:48,45-47:48,64:48
XTerm*cutNewLine:     off
XTerm*cutToBeginningOfLine: off
XTerm*titeInhibit:    on
XTerm*ttyModes:       intr ^c erase ^? kill ^u
XLoad*Background:     gold
XLoad*Foreground:     red
XLoad*highlight:     black
XLoad*borderWidth:    0
hpterm*Geometry:     80x65-0-0
hpterm*Background:    rgb:5b/76/86
hpterm*Foreground:    white
hpterm*Cursor:       white
hpterm*BorderColor:   white
hpterm*Font:         6x10
```

If these resources were stored in a file called `.Xdefaults` in your home directory, they could be added to any existing resources in the server with the following command:

```
$ xrdp -merge $HOME/.Xdefaults
```

This is frequently how user-friendly startup scripts merge user-specific defaults into any site-wide defaults. All sites are encouraged to set up convenient ways of

A-22 Reference

X Windows

automatically loading resources. See the Xlib manual section “Resource Manager Functions” for more information.

Examples

The following is a collection of sample command lines for some of the more frequently used commands. For more information on a particular command, please refer to that command’s manual page.

```
$ xrdp $HOME/.Xdefaults
$ xmodmap -e "keysym BackSpace = Delete"
$ mkfontdir /usr/local/lib/X11/otherfonts
$ xset fp+ /usr/local/lib/X11/otherfonts
$ xmodmap $HOME/.keymap.km
$ xsetroot -solid 'rgb:0.8/0.8/0.8'
$ xset b 100 400 c 50 s 1800 r on
$ xset q
$ mwm
$ xclock -geometry 48x48-0+0 -bg blue -fg white
$ xlsfonts '*helvetica*'
$ xwininfo -root
$ xhost -joesworkstation
$ xwd | xwd
$ xterm -geometry 80x66-0-0 -name myxterm $*
```

A

X Windows

Diagnostics

A wide variety of error messages are generated from various programs. The default error handler in Xlib (also used by many toolkits) uses standard resources to construct diagnostic messages when errors occur. The defaults for these messages are usually stored in `/usr/lib/X11/XErrorDB`. If this file is not present, error messages will be rather terse and cryptic.

When the X Toolkit Intrinsic encounters errors converting resource strings to the appropriate internal format, no error messages are usually printed. This is convenient when it is desirable to have one set of resources across a variety of displays (e.g. color vs. monochrome, lots of fonts vs. very few, etc.), although it can pose problems for trying to determine why an application might be failing. This behavior can be overridden by setting the `StringConversionsWarning` resource.

To force the X Toolkit Intrinsic to always print string conversion error messages, the following resource should be placed in the `.Xdefaults` file in the user's home directory. This file is then loaded into the `RESOURCE_MANAGER` property using the `xrdb` program:

```
*StringConversionWarnings: on
```

To have conversion messages printed for just a particular application, the appropriate instance name can be placed before the asterisk:

```
xterm*StringConversionWarnings: on
```

See Also

`bdfpcf(1)`, `bitmap(1)`, `fs(1)`, `hpterm(1)`, `mkfontdir(1)`, `mwm(1)`, `xauth(1)`, `xclock(1)`, `xcmsdb(1)`, `xfd(1)`, `xhost(1)`, `xinitcolor(1)`, `xload(1)`, `xlsfonts(1)`, `xmodmap(1)`, `xpr(1)`, `xprop(1)`, `xrdb(1)`, `xrefresh(1)`, `xset(1)`, `xsetroot(1)`, `xterm(1)`, `xwd(1)`, `xwininfo(1)`, `xwud(1)`, `Xserver(1)`, *Xlib - C Language X Interface*, and *X Toolkit Intrinsic - C Language Interface*.

A

A-24 Reference

Copyright

The following copyright and permission notice outlines the rights and restrictions covering most parts of the core distribution of the X Window System from MIT. Other parts have additional or different copyrights and permissions; see the individual source files.

Copyright 1984, 1985, 1986, 1987, 1988, 1989, 1990, 1991 by the Massachusetts Institute of Technology.

Permission to use, copy, modify, distribute, and sell this software and its documentation for any purpose is hereby granted without fee, provided that the above copyright notice appear in all copies and that both that copyright notice and this permission notice appear in supporting documentation, and that the name of MIT not be used in advertising or publicity pertaining to distribution of the software without specific, written prior permission. MIT makes no representations about the suitability of this software for any purpose. It is provided "as is" without express or implied warranty.

Trademarks

X Window System is a trademark of MIT.

Authors

A cast of thousands, literally. The MIT Release 5 distribution is brought to you by the MIT X Consortium. The names of all people who made it a reality will be found in the individual documents and source files. The staff members at MIT responsible for this release are: Donna Converse (MIT X Consortium), Stephen Gildea (MIT X Consortium), Susan Hardy (MIT X Consortium), Jay Hersh (MIT X Consortium), Keith Packard (MIT X Consortium), David Sternlicht (MIT X Consortium), Bob Scheifler (MIT X Consortium), and Ralph Swick (Digital/MIT Project Athena).

X Server

Synopsis

`X` : *displaynumber* [-*option*] *ttyname*

Description

“X” is the generic name for the window system server. It is started by the `dtlogin(1X)` program which is typically run by `init(1M)`. Alternatively it may be started from the `xinit(1)` program, which is called by `x11start`. The *displaynumber* argument is used by clients in their `DISPLAY` environment variables to indicate which server to contact (machines may have several displays attached). This number can be any number. If no number is specified, 0 is used. This number is also used in determining the names of various startup files. The *ttyname* argument is passed in by `init` and isn’t used.

The Hewlett-Packard server has support for the following protocols:

TCP/IP	The server listens on port $6000+n$, where n is the display number.
Local Socket IPC Mechanism	The socket file name is “ <code>/usr/spool/sockets/X11/*</code> ”, where “ <code>*</code> ” is the display number.
Shared Memory IPC	This is the default connection that the X Library will use to connect to an X server on the same machine if the <code>DISPLAY</code> environment variable is set to “ <code>local:*</code> ” or “ <code>:*</code> ” where “ <code>*</code> ” is the number of the display.

When the server starts up, it takes over the display. If you are running on a workstation whose console is the display, you cannot log into the console while the server is running.

A

Options

The following options can be given on the command line to the X server.

<code>-a <i>number</i></code>	Sets pointer acceleration (i.e. the ratio of how much is reported to how much the user actually moved the pointer).
<code>-audit <i>level</i></code>	Sets the audit trail level. The default level is 1, meaning only connection rejections are reported. Level 2 additionally reports all successful connections and disconnects. Level 0 turns off the audit trail. Audit lines are sent as standard error output.
<code>-auth <i>authorization-file</i></code>	Specifies a file which contains a collection of authorization records used to authenticate access.
<code>bc</code>	Disables certain kinds of error checking, for bug compatibility with previous releases (e.g., to work around bugs in R2 and R3 <code>xterms</code> and toolkits). Deprecated.
<code>-bs</code>	Disables backing store support on all screens.
<code>-c</code>	Turns off key-click.
<code>c <i>volume</i></code>	Sets key-click volume (allowable range: 0-100).
<code>-co <i>filename</i></code>	Sets name of RGB color database.
<code>-core</code>	Causes the server to generate a core dump on fatal errors.
<code>-dpi <i>resolution</i></code>	Sets the resolution of the screen, in dots per inch. To be used when the server cannot determine the screen size from the hardware.
<code>-f <i>volume</i></code>	Sets beep (bell) volume (allowable range: 0-100).
<code>-fc <i>cursorFont</i></code>	Sets default cursor font.
<code>-fn <i>font</i></code>	Sets the default font.
<code>-fp <i>fontPath</i></code>	Sets the search path for fonts. This path is a comma-separated list of directories which the server searches for font databases.

X Server

<code>-help</code>	Prints a usage message.
<code>-I</code>	Causes all remaining command line arguments to be ignored.
<code>-logo</code>	Turns on the X Window System logo display in the screen-saver. There is currently no way to change this from a client.
<code>nologo</code>	Turns off the X Window System logo display in the screen-saver. There is currently no way to change this from a client.
<code>-p <i>minutes</i></code>	Sets screen-saver pattern cycle time in minutes.
<code>-pn</code>	Allows X server to run even if one or more communications mechanisms fails to initialize.
<code>-pn</code>	Permits the server to continue running if it fails to establish all of its well-known sockets, but establishes at least one.
<code>-r</code>	Turns off keyboard auto-repeat.
<code>r</code>	Turns on keyboard auto-repeat.
<code>-s <i>minutes</i></code>	Sets screen-saver timeout time in minutes.
<code>-su</code>	Disables save under support on all screens.
<code>-t <i>number</i></code>	Sets pointer acceleration threshold in pixels (i.e. after how many pixels pointer acceleration should take effect).
<code>-terminate</code>	Causes the server to terminate at server reset, instead of continuing to run.
<code>-to <i>seconds</i></code>	Sets default connection timeout in seconds.
<code>-tst</code>	Disables all testing extensions (e.g., <code>XTEST</code> , <code>XTrap</code> , <code>XTestExtension1</code>).
<code>ttyxx</code>	Ignored; for servers started the ancient way (from <code>init</code>).
<code>-terminate</code>	Causes server to terminate when all clients disconnect.
<code>v</code>	Sets video-on screen-saver preference. A window that changes regularly will be used to save the screen.

A

A-28 Reference

X Server

- v** Sets video-off screen-saver preference. The screen will be blanked to save the screen.
- wm** Forces the default backing-store of all windows to be **WhenMapped**; a less-expensive way of getting backing-store to apply to all windows.

You can also have the X server connect to **xdm(1)** or **dtlogin(1X)** using XDMCP. Although this is not typically useful as it doesn't allow **xdm** to manage the server process, it can be used to debug XDMCP implementations, and serves as a sample implementation of the server side of XDMCP. The following options control the behavior of XDMCP:

- query *host-name*** Enable XDMCP and send Query packets to the specified host.
- broadcast** Enable XDMCP and broadcast BroadcastQuery packets to the network. The first responding display manager will be chosen for the session.
- indirect *host-name*** Enable XDMCP and send IndirectQuery packets to the specified host.
- port *port-num*** Use an alternate port number for XDMCP packets. Must be specified before any **-query**, **-broadcast** or **-indirect** options. Default port number is 177.
- class *display-class*** XDMCP has an additional display qualifier used in resource lookup for display-specific options. This option sets that value, by default it is "MIT-Unspecified" (not a very useful value).
- cookie *xdm-auth-bits*** When testing **XDM-AUTHENTICATION-1**, a private key is shared between the server and the manager. This option sets the value of that private data (not that it's very private, being on the command line and all ...).
- displayID *display-id*** Yet another XDMCP-specific value, this one allows the display manager to identify each display so that it can locate the shared key.

A

X Server

Running From `init`

Though X will usually be run by `dtlogin` from `init`, it is possible to run X directly from `init`. For information about running X from `dtlogin`, see the `dtlogin` man page.

To run X directly from `init`, it is necessary to modify `/etc/inittab` and `/etc/gettydefs`. Detailed information on these files may be obtained from the `inittab(4)` and `gettydefs(4)` man pages.

To run X from `init` on display 0, with a login `xterm` running on `/dev/ttypf`, in `init` state 3, the following line must be added to `/etc/inittab`:

```
X0:3:respawn:env PATH=/bin:/usr/bin/X11:/usr/bin xinit -L ttyqf -- :0
```

To run X with a login `hpterm`, the following should be used instead:

```
X0:3:respawn:env PATH=/bin:/usr/bin/X11:/usr/bin xinit hpterm =+1+1 -n \  
login -L ttyqf -- :0
```

In addition, the following line must be added to `/etc/gettydefs` (this should be a single line):

```
Xwindow# B9600 HUPCL PAREN# CS7 # B9600 SANE PAREN# CS7 ISTRIP IXANY TAB3 #X login: #Xwindow
```

There should not be a `getty` running against the display whenever X is run from `xinit`.

A

A-30 Reference

Security

The sample server implements a simplistic authorization protocol, `MIT-MAGIC-COOKIE-1` which uses data private to authorized clients and the server. This is a rather trivial scheme; if the client passes authorization data which is the same as the server has, it is allowed access. This scheme is inferior to host-based access control mechanisms in environments with unsecure networks as it allows any host to connect, given that it has discovered the private key. But in many environments, this level of security is better than the host-based scheme as it allows access control per-user instead of per-host.

In addition, the server provides support for a DES-based authorization scheme, `XDM-AUTHORIZATION-1`, which is more secure (given a secure key-distribution mechanism), but as DES is not generally distributable, the implementation is missing routines to encrypt and decrypt the authorization data. This authorization scheme can be used in conjunction with XDMCP's authentication scheme, `XDM-AUTHENTICATION-1` or in isolation.

The authorization data is passed to the server in a private file named with the `-auth` command line option. Each time the server is about to accept the first connection after a reset (or when the server is starting), it reads this file. If this file contains any authorization records, the local host is not automatically allowed access to the server, and only clients which send one of the authorization records contained in the file in the connection setup information will be allowed access. See the `Xau` manual page for a description of the binary format of this file. Maintenance of this file, and distribution of its contents to remote sites for use there, is left as an exercise for the reader.

The sample server also uses a host-based access control list for deciding whether or not to accept connections from clients on a particular machine. This list initially consists of the host on which the server is running as well as any machines listed in the file `/etc/Xn.hosts`, where `n` is the display number of the server. Each line of the file should contain an Internet hostname (e.g., `expo.lcs.mit.edu`). There should be no leading or trailing spaces on any lines. For example:

```
joesworkstation
corporate.company.com
```

A

X Server

Users can add or remove hosts from this list and enable or disable access control using the `xhost` command from the same machine as the server. For example:

```
$ xhost +janesworkstation
janesworkstation being added to access control list
$ xhost +
all hosts being allowed (access control disabled)
$ xhost -
all hosts being restricted (access control enabled)
$ xhost
access control enabled (only the following hosts are allowed)
joesworkstation
janesworkstation
corporate.company.com
```

Signals

The X server attaches special meaning to the following signals:

- SIGHUP** This signal causes the server to close all existing connections, free all resources, and restore all defaults. It is sent by the display manager (`xdm` or `dtlogin`) whenever the main user's main application exits to force the server to clean up and prepare for the next user.
- SIGTERM** This signal causes the server to exit cleanly.
- SIGUSR1** This signal is used quite differently from either of the above. When the server starts, it checks to see if it has inherited `SIGUSR1` as `SIG_IGN` instead of the usual `SIG_DFL`. In this case, the server sends a `SIGUSR1` to its parent process after it has set up the various connection schemes. `xdm` uses this feature to recognize when connecting to the server is possible.

A

A-32 Reference

Fonts

Fonts are usually stored as individual files in directories. The list of directories in which the server looks when trying to open a font is controlled by the font path. Although most sites will choose to have the server start up with the appropriate font path (using the `-fp` option mentioned above), it can be overridden using the `xset` program.

Font databases are created by running the `mkfontdir` or `stmkdirs` program in the directory containing the compiled versions of the fonts (`mkfontdir`) or font outlines (`stmkdirs`.) Whenever fonts are added to a directory, `mkfontdir` or `stmkdirs` should be rerun so that the server can find the new fonts. If `mkfontdir` or `stmkdirs` is not run, the server will not be able to find any of the new fonts in the directory.

In addition, the X server supports font servers. A font server is a networked program that supplies fonts to X servers and other capable programs. In order to communicate with a font server, the font servers address must be supplied as part of the X server's font path. A font server's address is specified as:

transport/hostname:port-number

where *transport* is always "tcp", *hostname* is the hostname of the machine being connected to (no hostname means a local connection) and *port-number* is the tcp address that the font server is listening at (typically 7000.)

Diagnostics

Too numerous to list them all. If run from `init(1M)`, errors are logged in the file `/usr/adm/X*msgs`.

X Server

Files

<code>/etc/inittab</code>	Script for the <code>init</code> process
<code>/etc/gettydefs</code>	Speed and terminal settings used by <code>getty</code>
<code>/etc/X*.hosts</code>	Initial access control list
<code>/usr/lib/X11/fonts</code>	Top level font directory
<code>/usr/lib/X11/rgb.txt</code>	Color database
<code>/usr/lib/X11/rgb.pag</code>	Color database
<code>/usr/lib/X11/rgb.dir</code>	Color database
<code>/usr/spool/sockets/X11/*</code>	IPC mechanism socket
<code>/usr/adm/X*msgs</code>	Error log file
<code>/usr/lib/X11/X*devices</code>	Input devices used by the server. This file contains many example configurations.
<code>/usr/lib/X11/X*screens</code>	Screens used by the server. This file contains many example configurations.
<code>/usr/lib/X11/X*pointerkeys</code>	Keyboard pointer device file. This file contains many example configurations.
<code>/usr/lib/X11/XHPkeymaps</code>	Key device database used by the X server.

Notes

The option syntax is inconsistent with itself and `xset(1)`. The acceleration option should take a numerator and a denominator like the protocol. The color database is missing a large number of colors. However, there doesn't seem to be a better one available that can generate RGB values.

Copyright

Copyright 1984-1997 Massachusetts Institute of Technology.

Copyright 1992-1997 Hewlett Packard Company.

See `X(1)` for a full statement of rights and permissions.

A-34 Reference

Origin

MIT Distribution.

See Also

dtlogin(1X), bdfpcf(1), fs(1), getty(1M), gettydefs(4), gwindstop(1), hpterm(1), init(1M), inittab(4), mkfontdir(1), rgb(1), stmkdirs(1), x11start(1), xclock(1), xfd(1), xhost(1), xinit(1), xinitcolormap(1), xload(1), xmodmap(1), xrefresh(1), xseethru(1), xset(1), xsetroot(1), xterm(1), xwcreate(1), xwd(1), xwdestroy(1), xwininfo(1), xwud(1).



Index

3

3BitCenterColor, 3-11

B

BufferSwapsonVerticalBlank, 3-3

buttons

- actions, 5-13
- changing mappings, 5-14
- coding, 5-20
- mappings, 5-14
- mouse, 5-13, 5-17
- operation functions, 5-18

C

CenterColor, 3-11

chording, 5-20

colormap, 5-12

ColormapManagement, 3-21

ColormapsandColormapManagement,
3-21

ColorRecovery, 3-7

Compiling

- HP PEX, 2-8
- HP-PHIGS, 2-5
- Starbase, 2-2

configuration

- special, 5-1

Configurations

- XServer, 3-16

CRX, 3-25

CRX24Z, 3-26

CRX48Z, 3-28

D

DBEDoubleBufferExtension, 3-3

default

- screen configuration file, 5-1
- X0devices configuration, 5-15

device

- input, 5-3

Devicedriver

- Libraries, 2-7

device drivers, 5-5

DeviceFiles, 3-13

DIN interface, 5-3

DoubleBufferExtensionDBE, 3-3

DualCRX, 3-25

Dvorak keyboard, 5-27

DynamicLoading, 3-9

E

EnableOverlayTransparency, 3-11

EnvironmentVariables

- Obsolete, 3-12
- SettingUnsetting, 3-2

/etc/hosts file, 5-2

F

File

- Device, 3-13

- Xscreens, 3-1

Freedom Series™, 3-41

G

GLX, 3-41
GraphicsResourceManagerGRM, 3-9
GRMGraphicsResourceManager, 3-9
GRX, 3-25

H

HCRX, 3-30
host
 names, 5-2
HPCOLORRecovery, 3-7
HP-HIL interface, 5-3, 5-7
HP PEX
 Compiling, 2-8
HP-PHIGS
 Compiling, 2-5
HPVUEandSingleLogicalScreen, 3-7

I

ImageTextViaBitMap, 3-11
input devices, 5-3
IntegratedColorGraphics, 3-21
interfaces, input, 5-3
InternalColorGraphics, 3-25
InternalGrayScaleGraphics, 3-25

K

key
 bindings, 5-24
 map, 5-28
 remapping expressions, 5-25
keyboard, 5-17
 46021, 5-24, 5-29
 assigning mouse functions, 5-17
 C1429, 5-24, 5-29
 Dvorak, 5-27
 input, 5-24
 modifier keys, 5-21

Index-2**L**

Libraries
 Devicedriver, 2-7
Loading
 Dynamic, 3-9
loopback address, 5-2

M

mapping
 mouse buttons, 5-14
MBX, 3-4
modifier
 key bindings, 5-24
 keys, 5-21
modifying
 X0pointerkeys, 5-16
mouse
 button, 5-14
 button mappings, 5-14
 buttons, 5-13, 5-17
 functions, 5-17
 keys, 5-20
 without, 5-15
mouseless operation, 5-15
MultiDisplaySupport, 3-17
MultiScreenSupport, 3-18

O

ObsoleteEnvironmentVariables, 3-12
operation functions, 5-18

P

PEX
 Compiling, 2-8
PHIGS
 Compiling, 2-5
pointer
 direction keys, 5-17
 distance functions, 5-18
 movement functions, 5-17
 specifying keys, 5-22

printing
 key map, 5-28

R

remapping, 5-24, 5-25
reset functions, 5-19
RS-232C interface, 5-3, 5-5

S

screen
 default configuration file, 5-1
SettingUnsettingEnvironmentVariables,
 3-2
SharedMemory, 3-9
SingleLogicalScreenandHPVUE, 3-7
SingleLogicalScreenSLS, 3-6
SLSSingleLogicalScreen, 3-6, 3-7
special configurations, 5-1
SpecialDeviceFiles, 3-13
special input devices, 5-3
Starbase
 Compiling, 2-2

T

tablet size, 5-21
threshold functions, 5-19
Transparency, 3-10

V

Variables
 Environment, 3-2
VerticalBlank
 BufferSwapson, 3-3
VRX, 3-42
VUEandSingleLogicalScreen, 3-7

X

X0devices, 5-4
X0devices file, 5-4, 5-15
X0.hosts file, 5-2
X0screens file, 5-2, 5-15, 5-16
XConfigurations, 3-16
X*devices file, 5-3, 5-4, 5-5, 5-8, 5-10,
 5-15
X*.hosts file, 5-1
X keyboard devices, 5-3
xmodmap client, 5-26, 5-27
X pointer devices, 5-3
X*pointerkeys, 5-20, 5-21
X*pointerkeys file, 5-12, 5-15, 5-16,
 5-21
X*screens file, 5-1, 5-4
XscreensFile, 3-1
X Toolkit warning, 5-12
X Window System, 5-11
 configuring, 5-9

