Graphics Administration Guide for HP-UX 10.20. After using this document for a while, please fill out the Reader Reply form. Or, just e-mail us your comments at muserlink@fc.hp.com

About this Guide

Graphics Administration Guide:
Information for Programmers, System Administrators, and End-Users
Copyright © June 1997, June 1998, June 2000 Hewlett-Packard Company HP 9000 Workstations

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: 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. June 2000. . . Edition 3. This document is valid for HP9000 workstations running HP-UX 10.20 along with the December 1999 Workstation ACE and Quality Pack.

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 and OpenGL.

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.

This document provides a more elegant solution. 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**: Locations of important files.
- **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:

mknod /usr/include

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 every case...

Emphasized words are in italic type.

. . .device is a **freen**. . .

New terms being introduced are in bold-faced type.

. . .the <device_id>. . .

Conceptual values are in italic type, enclosed in angle brackets. These items are not verbatim values, but are descriptors of the type of item it is, and the user should replace the conceptual item with whatever value is appropriate for the context.

Chapter 1: Pathnames

This chapter contains information on locating files that reside at some location in the 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 might 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.

Find the API you're looking for. 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.

Starbase

Generic Name	Location in the file system
<*common*>	/opt/graphics/common
<*dev*>	/dev
<*formatters*>	/opt/graphics/starbase/formatters
<*nls*>	/opt/graphics/common/lib/nls/msg/C
<*sb-demos*>	/opt/graphics/starbase/demo
<*sb-font*>	/opt/graphics/common/stroke
<*sb-font-info*>	/opt/graphics/common/stroke/font_info
<*sb-incl*>	/opt/graphics/starbase/include
<*sb-lib*>	/opt/graphics/common/lib
<*sb-utils*>	/opt/graphics/starbase/demos/starbase/SBUTILS
<*screen*>	/dev/screen
<*starbase*>	/opt/graphics/starbase
<*tmp*>	/var/tmp
<*vue-config*>	/etc/vue/config
<*x11*>	/usr/lib/X11
<*x11-admin*>	/etc/X11
<*x11r6*>	/usr/lib/X11R6
<*x11r6-incl*>	/usr/include/X11R6
<*xconfig*>	/etc/X11

HP-PHIGS

Generic Name	**Location in the file system**
<*app-defaults*>	/usr/lib/X11/app-defaults
<*common*>	/opt/graphics/common
<*dev*>	/dev
<*nls*>	/opt/graphics/common/lib/nls/msg/C
<*phigs*>	/opt/graphics/phigs
<*phigs-demos*>	/opt/graphics/phigs/demos
<*phigs-examples*>	/opt/graphics/phigs/examples
<*phigs-incl*>	/opt/graphics/phigs/include
<*phigs-lib*>	/opt/graphics/phigs/lib
<*phigs-widget*>	/opt/graphics/phigs/include/Motif1.2
<*screen*>	/dev/screen
<*spool*>	/var/spool
<*starbase*>	/opt/graphics/starbase
<*vue-config*>	/etc/vue/config
<*x11*>	/usr/lib/X11
<*x11r6*>	/usr/lib/X11R6
<*x11r6-incl*>	/usr/include/X11R6
<*xconfig*>	/etc/X11

HP PEX

Generic Name	Location in the file system
<app-defaults>	/usr/lib/X11/app-defaults
<cge-examples>	/opt/graphics/PEX5/examples/cge
<cge-utils>	/opt/graphics/PEX5/utilities/cge
<contrib>	/opt/graphics/PEX5/contrib
<err-help>	/opt/graphics/PEX5/help5.1
<extensions>	/opt/graphics/PEX5/newconfig/usr/lib/X11/extensions
<hp-examples>	/opt/graphics/PEX5/examples/hp
<man>	/opt/graphics/PEX5/share/man
<nls>	/opt/graphics/PEX5/lib/nls/msg/C
<ora-examples>	/opt/graphics/PEX5/examples/OReilly
<pex>	/opt/graphics/PEX5
<pexd>	/opt/graphics/PEX5/lbin
<pex-examples>	/opt/graphics/PEX5/examples
<pex-fonts>	/opt/graphics/PEX5/fonts
<pex-incl>	/opt/graphics/PEX5/include/X11R6/X11/PEX5
<pex-lib>	/opt/graphics/PEX5/lib
<pex-utils>	/opt/graphics/PEX5/utilities
<profile>	/opt/graphics/PEX5/contrib
<rel-notes>	/opt/graphics/PEX5/newconfig/opt/graphics/PEX5
<spool>	/var/spool
<vhelp>	/opt/graphics/PEX5/help5.1
<vue>	/usr/vue
<x11>	/opt/graphics/PEX5/newconfig/usr/lib/X11
<x11-incl>	/usr/include/X11R6/X11
<x11r6>	/opt/graphics/PEX5/lib/X11R6
<x11r6-incl>	/usr/include/X11R6

Chapter 2: Compiling Your Application

This chapter provides information for compiling your 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. See Chapter 1 for details.

Compiling Starbase Applications

Compiling with Shared Libraries

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.

cc example.c –I<sb-incl> -L<common> -L<sb-lib> -lXwindow \ -lhpgfx -lXhp11 -lX11 -lm -ldld -o example

For FORTRAN:

fort77 example.f –I<sb-incl> -L<common> -L<sb-lib> -lXwindow \ -lhpgfx -lXhp11 -lX11 -lm -ldld -o example

For Pascal:

pc example.p –I<sb-incl> -Wl,-L<common> -Wl,-L<sb-lib> \ -lXwindow -lhpgfx -lXhp11 -lX11 -lm -ldld -o example

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. Or (preferred because of fewer side-effects), "-Wl,-a,archive".

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

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.

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

cc example.c –I<sb-incl> -Wl,-a,archive –L<common> -L<sb-lib> \ -ldd<device_driver> -lXwindow -lhpgfx1 -lhpgfx2 -lXhp11 \ -lXext -lX11 -Wl,-E -Wl,+n -l:libdld.sl -lm -o example

For FORTRAN, use:

fort77 example.f –I<sb-incl> -Wl,-a,archive –L<common> \ -L<sb-lib>-ldd<device_driver> -lXwindow -lhpgfx1 -lhpgfx2 \ -lXhp11 -lXext -lX11 -Wl,-E -Wl,+n -l:libdld.sl -lm \ -o example

For Pascal, use:

pc example.p –I<sb-incl> -Wl,-a,archive -Wl,-L<common> \ -Wl,-L<sb-lib> -ldd<device_driver> -lXwindow -lhpgfx1 \ -lhpgfx2 -lXhp11 -lXext -lX11 -Wl,-E -Wl,+n \ -l:libdld.sl -lm -o example

Compiling HP-PHIGS Applications

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<phigs-incl> -L<common>/lib –L<phigs-lib> \-I<phigs-widget>/Motif1.2_R6 -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: Shared libraries are used by default unless you specify that you want to use archived libraries (by specifying "-Wl,-a,archive").

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

cc example.c –I<phigs-incl> -Wl,-a,archive –L<common>/lib \ -L<phigs-lib> -I<phigs-widget>/Motif1.2_R6 –ldddl<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<phigs-incl> -Wl,-a,archive –L<common>/lib \ -L<phigs-lib> -I<phigs-widget>/Motif1.2_R6 -ldddl -lddgcrx \ -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.

Device Driver Libraries

The following tables list the device driver libraries that should be used with particular devices.

CRX Family	
Output Device	**Link Line Options**
Integrated Color Graphics (Model 705, 710, 715/33, 715/50, 715/75, 725/50, 725/75) Internal Color Graphics (Model 712/60, 712/80, 712/80i, 712/100, 715/64, 715/80, 715/100, 715/100XC, 725/100) HP CRX/GRX HP Dual CRX HP CRX-24[Z]	libddgcrx.a or libddgcrx.sl
HP CRX-48Z	libddcrx48z.a or libddcrx48z.sl

HCRX and HP VISUALIZE Families	
Output Device	**Link Line Options**
HP VISUALIZE-EG HP HCRX-8[Z] HP HCRX-24[Z] HP VISUALIZE-8 HP VISUALIZE-24	libddhcrx.a or libddhcrx.sl
HP VISUALIZE-48[XP]	libddhcrx48.a or libddhcrx48.sl

HP VISUALIZE-FX Family	
Output Device	**Link Line Options**
Legacy APIs	
HP VISUALIZE-EG	libddhcrx.a or libddhcrx.sl
HP VISUALIZE-FX2 HP VISUALIZE-FX4 HP VISUALIZE-FX6	libddvisx.a or libddvisx.sl
HP VISUALIZE-FXE	libddfxe.a or libddfxe.sl
OpenGL	
HP VISUALIZE-EG	libddhcrx.sl
HP VISUALIZE-FX2 HP VISUALIZE-FX4 HP VISUALIZE-FX6	libddvisxgl.sl libddvmd.sl
HP VISUALIZE-FXE	libddfxegl.sl

HP-GL Plotters	
Output Device	**Link Line Options**
HP 7440A HP 7470A HP 7475A HP 7550A HP 7570A HP 7575A HP 7576A HP 7580A/B HP 7585B HP 7586B HP 7595A HP 7596A HP 9111A HP C1600A HP C1601A	libddhpgl.a libdvio.a or libddhpgl.sl libdvio.sl
HP 7510A HP 7550A HP 7570A HP 7575A HP 7576A HP 7580B[1] HP 7585B[1] HP 7586B HP 7595A/B HP 7596A/B HP 7599A HP C1600A HP C1601A HP C1602A[1] HP C1620A HP C1625A HP C1627A HP C1629A HP C1631A	libddCADplt.a or libddCADplt.sl
[1] With HP-GL/2 plug-in cartridge	

Miscellaneous Device Drivers	
Output Device	**Link Line Options**
Remote Rendering: Xlib Pixmap (VMX)	libddvmx.sl[1]
Remote Rendering: Xlib 2D protocol (SOX)	libddsox11.a or libddsox11.sl
Display List	libdddl.a or libdddl.sl
Computer Graphics Metafile (CGM) File Format	libddhpcgm.a or libddhpcgm.sl
Starbase Memory Driver (pixel-major packing orders)	libddSMDpix.a or libddSMDpix.sl
Starbase Memory Driver (plane-major packing order)	libddSMDpln.a or libddSMDpln.sl
The Personal Visualizer File Format	libddhpsbv.a or libddhpsbv.sl
[1]The ".a" version of the VMX driver is bundled with the hpgfx library.	

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: The PEX library is dependent on the math library.

A compile line will typically appear:

```
cc program.c -lPEX5 -lXext -lX11 -lm –ldld
```

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

Compiling OpenGL Applications

HP's implementation of OpenGL is supported on workstations with HP VISUALIZE-FX graphics.

To compile a program that does not use the OpenGL utilities, use a makefile that looks like this:

```
INCDIR= -I/opt/graphics/OpenGL/include
LIBDIR=-L/opt/graphics/OpenGL/lib
LIBS=-lGL -lXext -lX11 -lm -ldld

meow : meow.c
   c89 $(INCDIR) $(LIBDIR) -o meow meow.c $(LIBS)
```

To compile a program that does use the OpenGL utilities, use a makefile that looks like this:

```
INCDIR= -I/opt/graphics/OpenGL/include
LIBDIR=-L/opt/graphics/OpenGL/lib
LIBS=-lGLU -lGL -lXext -lX11 -lm -ldld

meow : meow.c
   c89 $(INCDIR) $(LIBDIR) -o meow meow.c $(LIBS)
```

Chapter 3: X Windows: HP-UX 10.20

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.

X Server Configuration

Configuration of the X server is supported through SAM via an icon titled "X Server Configuration". This icon resides either at SAM's top level or under the top-level "Display" icon. This location is determined by the version of the HP-UX operating system (later HP-UX releases will place "X Server Configuration" under the "Display" folder).

There are several X*screens files used to configure the operation of the X server. The SAM graphical user interface for X server configuration is provided to simplify complexity and facilitate ease of use. While it is still possible to modify these files manually (see below), using the SAM interface greatly simplifies the process for creating Multi-Display and Single Logical Screen configurations.

Our SAM component has the following actions:

Configure Print Server...
Modify Multi-Screen Layout...
Modify Server Options...
Single Logical Screen (SLS) ->

Describe Screen...
Identify Screen
Modify Default Visual...
Modify Screen Options...
Add Screen to Configuration
Remove Screen from Configuration

The first group of actions can be thought of as "global" actions. They will typically be active regardless of what has been selected. If any of these menu items is not visible, it is because it is not supported under the current configuration. For example, on systems containing only one graphics screen, the last three menu items will not be visible.

The second group of actions can be thought of as "screen" actions. They will be activated depending on which screens have been chosen. It is also possible that the last two actions (Add and Remove) will be absent. When only one graphics screen is present, SAM will treat this screen as though it is always configured. Preselecting both configured and unconfigured screens will result in only the first two screen menu options being active.

X*screens File

For manual changes, please refer to the sample files in the */etc/X11/* directory. Three files of particular interest are the *X0screens*, *X0devices*, and *X0pointerkeys* files.

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: 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.

The X*screens File Format

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

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 a 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 (for example, */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: 0,0,0

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. The 10.20 X Server patches shipped after July, 1997 and all 11.00 versions of the X Server perform delayed loading of the X extensions.

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 that are documented in a file in the X server information directory (for example, */usr/lib/X11/Xserver/info/screens/hp*).

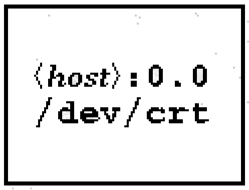
Sample X*screens Files

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
```

Figure 1: Results of Minimal Legal `X*screens` File

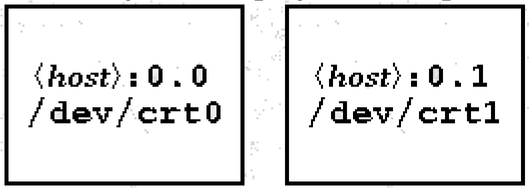


```
⟨host⟩:0.0
/dev/crt
```

- 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 "<host>:0.0" and "<host>:0.1", respectively.

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

Figure 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 to run 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. 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.

- 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.

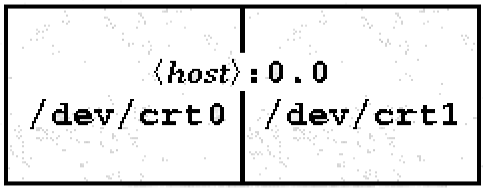
```
Screen /dev/crt
        ScreenOptions
                EnableOverlayTransparency
```

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

- 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
```

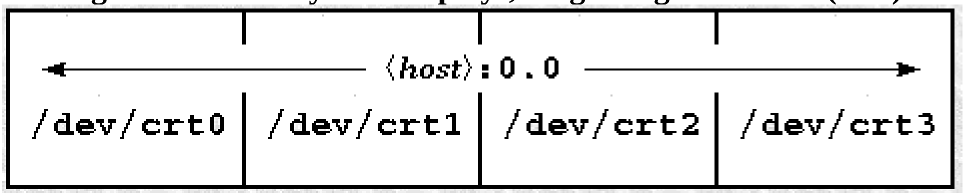
Figure 3: Two Physical Displays, Single Logical Screen (1X2)



- 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.

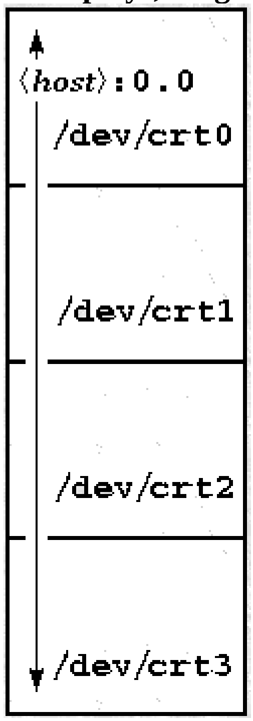
```
SingleLogicalScreen 1 4
        /dev/crt0 /dev/crt1 /dev/crt2 /dev/crt3
```

Figure 4: Four Physical Displays, Single Logical Screen (1X4)



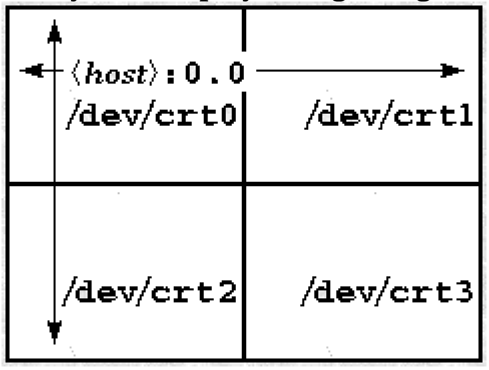
```
SingleLogicalScreen 4 1
        /dev/crt0
        /dev/crt1
        /dev/crt2
        /dev/crt3
```

Figure 5: Four Physical Displays, Single Logical Screen (4X1)



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

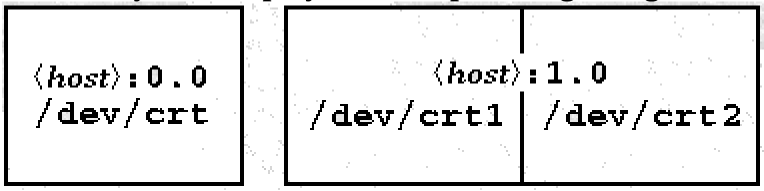
Figure 6: Four Physical Displays, Single Logical Screen (2X2)



- 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 7: Three Physical Displays, Screen plus Single Logical Screen (1X2)



Miscellaneous Topics

Double Buffer Extension (DBE)

DBE is an extension to the X server that provides a double-buffering Application Programming Interface (API).

Note: MBX (the **M**ulti-**B**uffering e**X**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: *MBX_SWAP_BUFFERS_ON_VBLANK* is obsolete with this release. The SwapBuffersOnVBlank Screen Option works for both DBE and MBX.

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[XP]
- HP VISUALIZE-FX^2
- HP VISUALIZE-FX^4
- HP VISUALIZE-FX^6
- HP VISUALIZE-FX^5 and FX^{10}
- HP VISUALIZE-FXE

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:

- HP VISUALIZE-EG
- HCRX-8[Z], HCRX-24[Z]
- HP VISUALIZE-8, HP VISUALIZE-24, and HP VISUALIZE-48[XP].
- HP VISUALIZE-FX^2, HP VISUALIZE-FX^4, HP VISUALIZE-FX^6
- HP VISUALIZE-FX^5 and FX^{10}
- HP VISUALIZE-FXE

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).

Power-Saving States Defined by VESA

Level State DPMS Compliance

Requirements	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.

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.

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.

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: *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.

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: *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[XP]
- HP VISUALIZE-FX^2
- HP VISUALIZE-FX^4
- HP VISUALIZE-FX^6
- HP VISUALIZE-FX^5 and FX^{10}
- HP VISUALIZE-FXE

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 particulary 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 no longer the default transport mechanism between the X server and client applications, and SMT is not an officially supported transport mechanism. However, it can be enabled (see "Enabling and Disabling of SMT") to achieve performance advantages.

SMT will be active between Xserver and client connections if the following are true:

- SMT is enabled (see "Enabling and Disabling of SMT").
- The client and server are actually on the same host.
- A Display Name of any of the forms listed below are used. ":0.0" is used for simplicity. This behavior is equally applicable for displays such as ":1.0".

  :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. 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: 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.

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. Such a specification in the X*screens file might appear as:

```
ServerOptions
SMTSizes 100000,90000,90000
```

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:

<region_size>[, *<high_water>* [, *<buffer_size>*]] with no embedded blanks, e.g.

32000,16000,5000
32000
0

Although the default is "*SMTSizes 0,0,0,*" a specification of "*SMTSizes 100000,90000,90000*" is optimal in most situations if SMT is desired.

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.

<region_size>

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) except for a diskless Xserver which has a default of 0.

<high_water>

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, make the value the 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.

Enabling and Disabling of SMT

The special value of 0 for Region Size specifies that SMT is to be disabled. Disabling SMT removes it from the Xserver causing it not to be available for any transports. Any non-zero Region Size enables SMT on the Xserver. The default Region Size for a diskless Xserver is zero (disabling SMT from the Xserver) to minimize the access of files across the network. The default Region Size for non-siskless Xserver is 100000 (which is rounded to 0x19000).

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 f*ork()* 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 such usages can be detected. Symptoms include reporting the error, and hanging applications.
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
- HP VISUALIZE-EG
- HCRX-8[Z], HCRX-24[Z]
- HP VISUALIZE-8, HP VISUALIZE-24, and HP VISUALIZE-48[XP]
- HP VISUALIZE-FX^2, HP VISUALIZE-FX^4, and HP VISUALIZE-FX^6

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

There are two components to the Color Recovery process. First, a different dither-cell size (16X2) 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 *DisableColorRecovery* Screen Option in the X*screens file before starting the server

---

Note: 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.

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.

Caution: 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.

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: 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
HP VISUALIZE-EG
HCRX-8[Z], HCRX-24[Z]
HP VISUALIZE-8, HP VISUALIZE-24, and HP VISUALIZE-48[XP]
HP VISUALIZE-FX^2, HP VISUALIZE-FX^4, and HP VISUALIZE-FX^6
HP VISUALIZE-FX^5 and FX^{10}
HP VISUALIZE-FXE

X*screens File Screen Option To Use:
*CountTransparentInOverlayVisua*l

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:
HP VISUALIZE-EG
HCRX-8[Z]
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.

X*screens File Screen Option To Use:

ImageTextViaBitMap

Note: Using this option will reduce text performance.

The ImageTextViaBitMap screen option is supported on all graphics devices supported on the HP 9000/700 machines, except the following:

PersonalVRX, TurboVRX
Freedom Series™ Graphics (S3150, S3250 and S3400)

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 by using SAM (the System Administration Manager tool):

- From SAM's main window, double-click "Peripheral Devices".
- From the "Peripheral Devices" window, double-click "Cards".
- A window will appear, containing a list of all cards that are in your machine. Once you select any of them (by single-clicking), the "Actions" menu will contain the options "Show Device Files" and "Create Device Files." Choose whichever option you desire.

Supported X Configurations

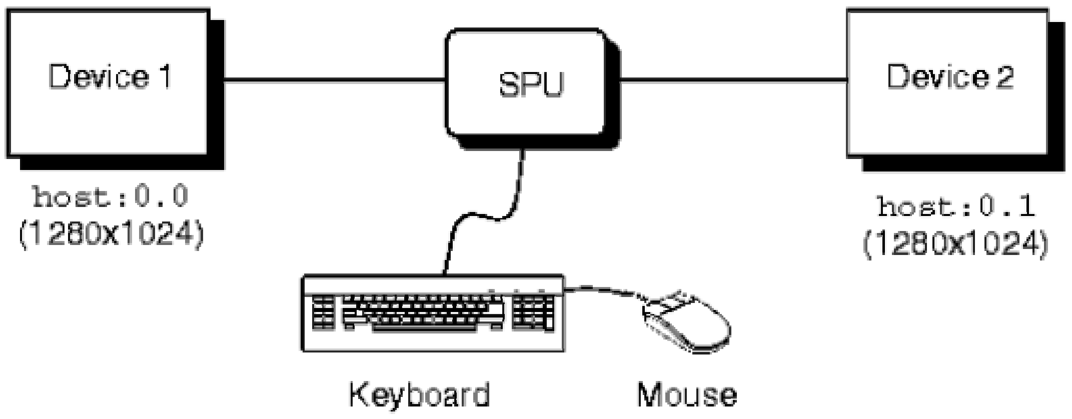
Multi-Display Support

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.

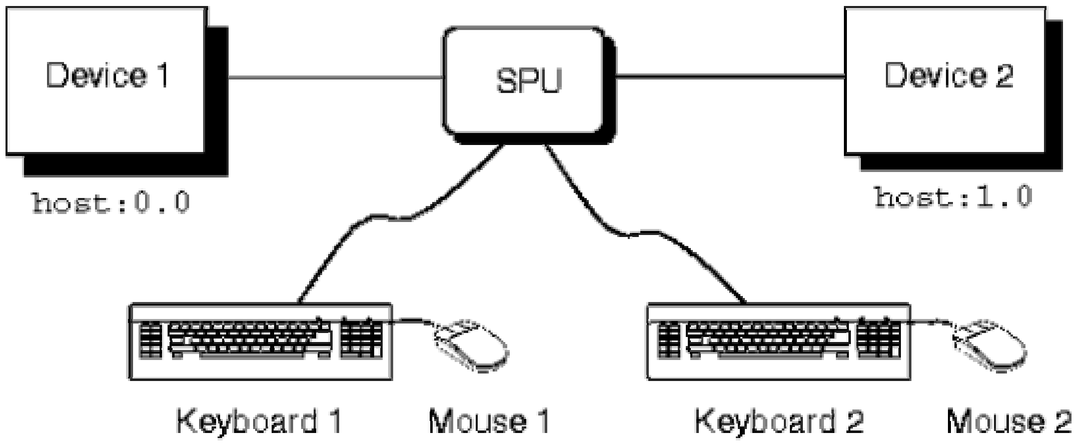
Multi-Screen
A configuration in which a single X server with a mouse and keyboard drives multiple graphics devices (where each display is a different X Screen) concurrently while only allowing the cursor, not windows, to be moved between displays.

Figure 8: Multi-Screen



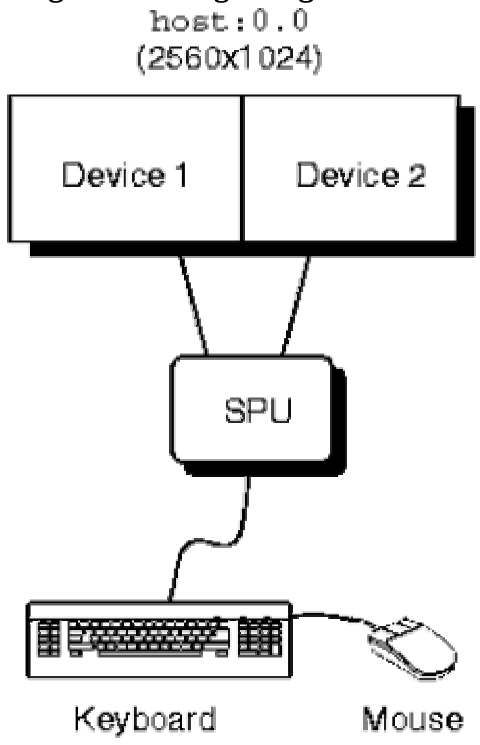
Multi-Seat
A configuration with multiple instantiations of the X server, each with its own mouse, keyboard, and display(s). Multi-seat is not currently supported in any HP-UX 10.* release.

Figure 9: Multi-Seat



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.

Figure 10: Single Logical Screen



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

Multi-Screen Support

The list of supported multi-display configurations is rather large, and it changes whenever a new graphics device is introduced. Thus, if you are considering a Single Logical Screen or any other multi-display configuration, we recommend consulting your HP Sales Representative and inquiring whether the configuration you have in mind is indeed supported.

There are general guidelines, however. For example:

- Multi-display configurations may be limited by available power. Depending on the capacity of your computer's power supply, and the power demands of the combination of graphics cards you are considering, there may or may not be enough power to operate them all.
- Single Logical Screen configurations must use identical graphics devices (see the next section).

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.

Note: The on-board and "card" versions of the same device can be considered identical; for example, you could use an on-board HP VISUALIZE-EG graphics device and an HP VISUALIZE-EG graphics card, and still consider them identical devices, thus permitting a 1X2 SLS or a 2X1 SLS.

SLS is enabled by using SAM (the System Administration Manager tool, /usr/sbin/sam). To enable an SLS configuration, start SAM, and follow the instructions below:

1. Double-click on the "X Server Configuration" button. A window entitled "Graphics" appears, containing an icon for every graphics device on your system.
2. Select the devices you want to combine into an SLS (click the mouse on the first device, and [Ctrl]-click on the others). At this point, all the devices you want to combine into an SLS configuration should be highlighted.
3. From the "Actions" menu, choose the menu item "Modify Multi-Screen Layout". A dialog box appears, allowing you to specify exactly how you want your SLS configuration to be.

Note: If your machine has only one graphics device, the "Modify Multi-Screen Layout" menu option does not even appear, since multiple devices cannot occur in a single-device context.

Note: DHA (Direct Hardware Access) is not supported in a window that spans multiple screens. This means, for example, that while graphics is supported to a window spanning two or more screens, accelerated graphics is not. "Spanning," in this context, includes a window that is two or more screens in size, as well as a window that is partially on one screen and partially on another (even though it would fit on a single screen if it were moved).

SLS can also be 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 nXm 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
```

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

Be sure 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.

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

Distributed Single Logical Screen (SLS/d)

SLS/d uses a Master/Slave relationship between X displays to create and manage the logical screen. SLS/d is functionally identical to SLS, except for the way the screens are configured. To configure an SLS/d screen, you must first choose a group of X displays, all running the same version of HP-UX 10.20 (running the January 1999 periodic patch or subsequent HP-UX 10.20 patch).

In addition, all of the systems running the displays must be running */usr/bin/X11/SLSd_daemon* (this is normally launched automatically when the system is booted; if it is not, you may start it by hand, as root).

Then, the master and slaves must be configured correctly. The SLS/d master X server can execute on any appropriate system, including one that is also executing a SLS/d slave X server. The SLS/d master X server does not require a graphics display device. The SLS/d slave displays must all have the same underlying graphics display device. To determine this, run "*/bin/graphinfo <device_file>*" (where *<device_file>* represents the graphics device file (e.g., */dev/crt0*) listed in that slave's */etc/X11/X*screens* file) and search for the "*graphics product*".

To Configure the SLS/d Slave X Servers

Configuring an SLS/d slave X server is identical to configuring a normal (non-SLS) screen as described above. The catch is that if any Screen or ServerOption(s) are configured for one slave, they must be configured for all of the slave X servers.

For example, to configure SLS/d slave X servers on *hpslsd1*, *hpslsd2*, *hpslsd3*, and *hpslsd4*, do the following as *root*:

- Copy each system's /etc/X11/X0screens file to /etc/X11/X50screens (the use of "50" as the display name is an SLS/d convention only; you can choose any number from 0-99 that meets your needs). Unless you're making changes to screen options or server options, you need not edit this file any further. This step can be accomplished using SAM by saving the default configuration to "X50screens".
- Create a file called /etc/X11/X50devices (unless you chose a different display number, in which case you would use the same number you selected for the X*screens file) and enter the following:

  ```
  first NULL keyboard
  first NULL pointer
  ```

- Make sure that SLSd_daemon is running on each system, then take the systems down to init state 2 (e.g., as *root*: */etc/init 2*).

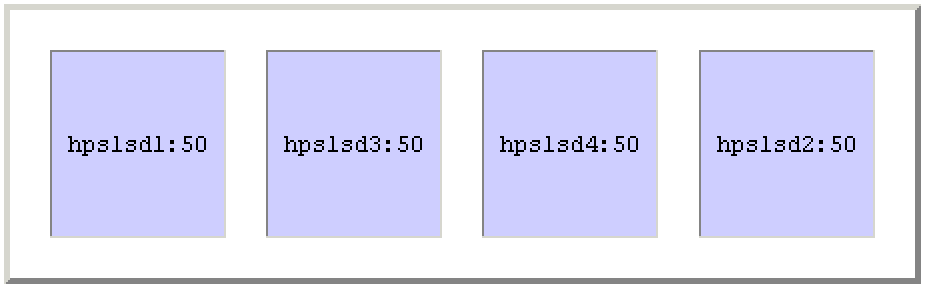
To Configure the SLS/d Master X Server

Configuring the SLS/d Master is almost identical to configuring a SLS screen as described above. As root, edit /etc/X11/X0screens on the system which will host the master (remember that the master can execute on the same system that is executing a slave; which is why we choose to use :50 for the slave display numbers).

Note: The use of "0" for the master is very important. If you change this number, CDE will not run properly unless you also make the necessary changes to have CDE run to a non-default screen location.

Then, to configure the logical screen as follows:

Figure 11: Logical Screen Configuration



Edit the */etc/X11/X0screens* file and enter:

```
SingleLogicalScreen 1 4
  hpslsd1:50 hpslsd3:50 hpslsd4:50 hpslsd2:50
```

Then, on the master system, make sure that SLSd_daemon is running and enter init state 4 (e.g., as root: /etc/init 4).
You will notice all of the slave X servers starting in turn, then finally when the master initializes, you will see your CDE environment start.

Up to 64 displays can be configured as a Distributed Single Logical Screen with the following restrictions:

- All SLS/d slave systems must be running HP-UX 10.20 with the January 1999 periodic patch or a subsequent HP-UX 10.20 patch;
- All SLS/d slave displays must be using the same graphics display device;
- All SLS/d slave displays must be in the same resolution (see */opt/graphics/common/bin/setmon*)
- The systems hosting the graphics devices (e.g., hpslsd1) need not be the same model. For example, hpslsd1 could be a J2240 and hpslsd2 could be a C360. They just need to be running the same graphics devices.

SLS/d is also used in 3DSLS/d, or 3D Distributed Single Logical Screen. In this case, you may also need to specify a fast LAN IP address.

Note: Only do this step if you have a gigabit ethernet connecting each slave to the master!

In addition, the fast LAN IP addresses must all be on the same subnet.

In the master's /etc/X11/X0screens file, modify the entries to also specify their Fast LAN IP addresses (those IP addresses associated with the Gigabit Ethernet). Assume that the following is true:

hpslsd1 has the fast LAN on 17.1.1.200
hpslsd2 has the fast LAN on 17.1.1.201
hpslsd3 has the fast LAN on 17.1.1.202

Then, to configure the SLS/d with fast LAN using a similar configuration as used above, modify the X0screens file as follows:

```
SingleLogicalScreen 1 4
  hpslsd1:50/17.1.1.200
  hpslsd3:50/17.1.1.201
  hpslsd4:50/17.1.1.202
  hpslsd4:50/17.1.1.203
```

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

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 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: 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. 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. 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.

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

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.

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 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.

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[XP] 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 [Dual] VISUALIZE-EG will refer to double-buffer mode only.

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[XP] is a two-board accelerated device that fills two slots. If you add either the optional texture-mapping memory card or the optional video-out card, it becomes a three-board set that fills three slots. Add both optional cards, and it becomes a four-board set, but it still fills only three slots. In any case, it 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 [Dual] VISUALIZE-EG 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[XP]:

- 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 Screen Options

The following Screen Options are supported:

- *CountTransparentInOverlayVisual*
- *DisableColorRecovery*
- *EnableOverlayTransparency* (HCRX-8[Z], HP [Dual] VISUALIZE-EG 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 [Dual] VISUALIZE-EG and HP VISUALIZE-8 Visuals and Double-Buffer Support

The eight-plane HCRX-8[Z], HP [Dual] VISUALIZE-EG 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 [Dual] VISUALIZE-EG 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 [Dual] VISUALIZE-EG'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).

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 [Dual] VISUALIZE-EG and HP VISUALIZE-8

The HCRX-8[Z], HP [Dual] VISUALIZE-EG 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 [Dual] VISUALIZE-EG 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 CountTransparentInOverlayVisual can be used to include the transparent index in the colormap size (256 entries instead of 255).

Programmers' 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.

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

The HCRX-24[Z], HP VISUALIZE-24, and HP VISUALIZE-48[XP] 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).

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, *pOVis;
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. */
        ...
        nOVisuals = numLongs/sizeof(OverlayVisualPropertyRec);
        pOVis = pOverlayVisuals;
        while (--nOVisuals >= 0)
           {
           if ( pOVis->transparentType == TransparentPixel )
              {/* Found a transparent overlay visual, set ident. aside. */};
           pOVis++;
           }
        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 [Dual] VISUALIZE-EG 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.

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[XP]: 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 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. 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. 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,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);
}
```

HP VISUALIZE-FX (FX^2, FX^4 and FX^6) Device-Dependent Information

This section includes information on the HP VISUALIZE-FX^2, HP VISUALIZE-FX^4 and HP VISUALIZE-FX^6 graphics devices:

- The HP VISUALIZE-FX^2 has 8 overlay planes, 24 image planes, a 24-bit Z buffer and 8 hardware colormaps.
- The HP VISUALIZE-FX^4 has 8 overlay planes, 48 image planes, a 24-bit Z buffer and 8 hardware colormaps.
- The HP VISUALIZE-FX^6 has 8 overlay planes, 48 image planes, 16 alpha planes, a 24-bit Z buffer and 8 hardware colormaps.

HP VISUALIZE-FX2/4/6 graphics devices contain 2D hardware acceleration similar to that in other HP VISUALIZE devices as well as 3D acceleration for geometry, lighting, and shading. Optional texture mapping acceleration is also available.

Supported Visuals

HP VISUALIZE-FX2/4/6 graphics devices support all of the following visuals:

- Class PseudoColor Depth 8 Layer Image
- Class PseudoColor Depth 8 Layer Overlay
- Class PseudoColor Depth 8 Layer Overlay Transparent
- Class TrueColor Depth 8 Layer Image
- Class PseudoColor Depth 12 Layer Image
- Class DirectColor Depth 12 Layer Image
- Class TrueColor Depth 12 Layer Image
- Class DirectColor Depth 24 Layer Image
- Class TrueColor Depth 24 Layer Image

The following visuals are enabled by default on the HP VISUALIZE-FX^2:

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

The default set of visuals on the HP VISUALIZE-FX^4 and HP VISUALIZE-FX^6 depend on the stereo mode setting.

In non-stereo mode, the following visuals are enabled by default on the HP VISUALIZE-FX^4 and HP VISUALIZE-FX^6:

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

In stereo mode, the following visuals are enabled by default on the HP VISUALIZE-FX4 and HP VISUALIZE-FX6:

- Class PseudoColor Depth 8 Layer Image -
  supports DBE hardware double-buffering
- Class PseudoColor Depth 8 Layer Overlay -
  supports DBE software double-buffering
- Class PseudoColor Depth 8 Layer Overlay Transparent –
  supports DBE software double-buffering
- Class TrueColor Depth 8 Layer Image -
  supports DBE hardware double-buffering
- Class DirectColor Depth 12 Layer Image -
  supports DBE hardware double-buffering
- Class TrueColor Depth 12 Layer Image -
  supports DBE hardware double-buffering
- Class DirectColor Depth 24 Layer Image -
  supports DBE hardware double-buffering
- Class TrueColor Depth 24 Layer Image -
  supports DBE hardware double-buffering

Note: When running xdpyinfo or calling the *XGetVisualInfo() Xlib* function, some extra duplicate visuals may appear in the visual list. These extra visuals are created on behalf of the OpenGL extension to X (GLX). If necessary, the extra visuals can be disabled using the *DisableGlxVisuals* screen option. See "Disabling the GLX Visuals" below for more information.

Supported Screen Options

The following Screen Options are supported:

- *CountTransparentInOverlayVisual*
- *ImageTextViaBitMap*
- *EnableIncludeInferiorsFix*
- *Enable12BitPseudoColorVisual*
- *DisableGlxVisuals*

The following additional screen options are supported on the HP VISUALIZE-FX2, HP VISUALIZE-FX4 (stereo mode) and HP VISUALIZE-FX6 (stereo mode):

- *Disable12BitDirectColorVisual*
- *Disable12BitTrueColorVisual*

The following additional screen options are supported on the HP VISUALIZE-FX4 (non-stereo mode) and HP VISUALIZE-FX6 (non-stereo mode):

- *Enable12BitDirectColorVisual*
- *Enable12BitTrueColorVisual*

HP VISUALIZE-FX Configuration Hints

Overlay Visuals and Overlay Transparency

HP VISUALIZE-FX2/4/6 devices have two visuals in the overlay planes, both depth-8 PseudoColor. The first (default) overlay visual has 256 entries per colormap and no transparency. The second overlay visual has 255 entries per colormap and supports transparency.

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

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

Disabling the GLX Visuals

The HP VISUALIZE-FX2/4/6 products are the first set of HP graphics devices that supports the OpenGL extension to X (GLX). If HP OpenGL is installed on an HP VISUALIZE-FX2/4/6 system, then the GLX extension offers new entry points for obtaining more information about X visuals. As part of offering extended visual information, some extra X visuals appear in the X visual list. The extra visuals are simply duplicates of visuals that would normally appear in the X visual list. In case that the extra visuals cause problems with applications, a screen option can be used to disable them.

To disable the GLX visuals, add the DisableGlxVisuals Screen Option to the X*screens file. For example:

```
Screen /dev/crt/
    ScreenOptions
        DisableGlxVisuals
```

HP VISUALIZE-FX2/4/6 Colormaps

HP VISUALIZE-FX2/4/6 devices have a total of 8 hardware colormaps. Two of the colormaps are dedicated to the overlay planes, and the remaining six colormaps are dedicated to the image planes.

Of the 2 overlay colormaps, one is permanently reserved for the default colormap. The other overlay colormap is available to applications.

Of the 6 image colormaps, two are reserved for the 12-bit PseudoColor visual. The other 4 image colormaps are available for applications using other image plane visuals.

Changing the Monitor Type

A configuration tool is available to change the monitor type on HP VISUALIZE-FX2/4/6 devices. Monitor resolution, refresh rate and stereo mode settings may be configured using the setmon tool. To change the monitor type, the setmon command can be executed directly or done through the SAM system administration tool.

The setmon executable is located at */opt/graphics/common/bin/setmon*. Under SAM this component is located under the top-level "Display" folder, next to the "X Server Configuration" icon.

Note: Changing the monitor type while the X server is running will necessitate killing and restarting the X server.

HP VISUALIZE-FXE, FX5 and FX10 Device-Dependent Information

Disable 12 Bit Direct Color Visual

This section includes information on the HP VISUALIZE-FXE, FX5 and FX10 graphics devices. The HP VISUALIZE-FXE/5/10 has 8 overlay planes, 48 image planes a 24-bit z buffer and 4 hardware colormaps.

HP VISUALIZE-FXE/5/10 graphics devices contain 2D hardware acceleration similar to that in other HP VISUALIZE devices, as well as 3D acceleration for lighting, shading and texture mapping.

Supported Visuals

HP VISUALIZE-FXE/5/10 graphics devices support all of the following visuals:

- *Class PseudoColor Depth 8 Layer Image*
- *Class PseudoColor Depth 8 Layer Overlay*
- *Class PseudoColor Depth8 Layer Overlay Transparent*
- *Class DirectColor Depth 24 Layer Image*
- *Class TrueColor Depth 24 Layer Image*

The following visuals are enabled by default on the HP VISUALIZE-FXE/5/10:

- Class PseudoColor Depth 8 Layer Image
  Supports DBE hardware double-buffering
- Class PseudoColor Depth 8 Layer Overlay
  Supports DBE software double-buffering
- Class PseudoColor Depth 8 Layer Overlay Transparent
  Supports DBE software double-buffering
- Class DirectColor Depth 24 Layer Image
  Does not support DBE hardware or software double-buffering
- Class TrueColor Depth 24 Layer Image
  Does not support DBE hardware or software double-bufferin

Note: When running xdpyinfo or calling the *XGetVisualInfo() Xlib* function, some extra duplicate visuals may appear in the visual list. These extra visuals are created on behalf of the OpenGL extension to X (GLX). If necessary, the extra visuals can be disabled using the DisableGLxVisuals screen option. See the "Disabling the GLX Visuals" section for more information.

Supported Screen Options

The following screen options are supported:

- *CountTransparentInOverlayVisual*
- *ImageTextViaBitMap*
- *EnableIncludeInferiorsFix*
- *DisableGlxVisuals*

HP VISUALIZE-FXE/5/10 Configuration Hints

Overlay Visuals and Overlay Transparency

HP VISUALIZE-FXE/5/10 devices have two visuals in the overlay planes, both depth-8 PseudoColor. The first (default) overlay visual has 256 entries per colormap and no transparency. The second overlay visual has 255 entries per colormap and supports transparency.

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.

Disabling the GLX Visuals

The HP VISUALIZE-FXE/5/10 products support the OpenGL extension to X (GLX). If HP OpenGL is installed on anHP VISUALIZE-FXE/5/10 system, then the GLX extension offers new entry points for obtaining more information about X visuals. As part of offering extended visual information, some extra X visuals appear in the X visual list. The extra visuals are simply duplicates of visuals that would normally appear in the X visual list. In case that the extra visuals cause problems with applications, a screen option can be used to disable them.

To disable the GLX visuals, add the DisableGlxVisualsScreen Option to the X*screens file.For example:

```
Screen /dev/crt/
        ScreenOption
           DisableGlxVisuals
```

HP VISUALIZE-FXE/5/10 Colormaps

HP VISUALIZE-FXE/5/10 devices have a total of 4 hardware colormaps. 2 of the colormaps are dedicated to the overlay planes. The remaining 2 colormaps are dedicated to the image planes.

Of the 2 overlay colormaps, one is permanently reserved for the default colormap. The other overlay colormap is available to applications.

Changing the Monitor Type

A configuration tool is available to change the monitor type on HP VISUALIZE-FX, HP VISUALIZE-EG, and HP VISUALIZE-FXE/5/10 devices. This tool permits users to change the monitor's refresh rate, frame buffer resolution, and frame buffer memory configuration (e.g., Stereo, Double Buffer), when the device supports multiple options. To change the monitor type, the setmon command can be executed directly or done through the SAM system administration tool.

The setmon executable is located at */opt/graphics/common/bin/setmon*. Under SAM this component is located under the top-level "Display" folder, next tothe "X Server Configuration" icon.

Note: Changing the monitor type while the X server is running will necessitate killing and restarting the X server.

Freedom Series Graphics Device-Dependent Information

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

Note: The Freedom Series is no longer supported as of HP-UX 10.20 Workstation ACE (July, 1997); 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 1280X1024 @ 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:

Alternative Supported Freedom Video Formats

Screen Option	Resolution	Description
ntsc_cvo	640X480	U.S. composite TV format with CVO*
pal_cvo	768X576	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:

Alternative Unsupported Freedom Video Formats

Screen Option	Resolution	Description
ntsc	640X480	U.S. composite TV format
pal	768X576	European composite TV format
hdtv	1920X1024	60 Hz interlaced
stereo1	640X512	60 Hz frame rate per eye
stereo2	1280X512	60 Hz frame rate per eye
vga	640X480	Standard VGA
video60	1280X1024	60 Hz
video76	1280X1024	76 Hz

VRX Device-Dependent Information

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

Note: The PersonalVRX and the TurboVRX are no longer supported as of the February, 1999 10.20 Xserver cumulative patch; 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)

None of these visuals support DBE or 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 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
```

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
```

Chapter 4: 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 hpddddd, you would want the following lines.

```
hpaaaaa
hpccccc
hpddddd
```

Note: 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.

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.

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 multile 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.

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/tty0p0 as the X pointer:

```
Begin_Device_Description
Name      spaceball.1
Path      /dev/tty0p0
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.

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:

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	

[1]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.

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: 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/hi*l. 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.
#<comment>
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.

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 X0pointerkeys 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 choses 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.

Customizing the Mouse and Keyboard

This section describes the following customizations:

- Changing mouse button actions.
- The *xmodmap* client.
- Going mouseless.
- Customizing keyboard input.

Changing Mouse Button Actions

Normally, the mouse pointer buttons are mapped as follows:

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.

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

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)

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*pointerkey*s. 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>]

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:

ointer Movement Functions

Movement Option	**Function**	**Default Key**
Move the pointer to the left.	pointer_left_key	keypad_1
Move the pointer to the right.	pointer_right_key	keypad_3
Move the pointer up.	pointer_up_key	keypad_5
Move the pointer down.	pointer_down_key	keypad_2
Add a modifier key to the pointer direction keys.	pointer_key_mod1	(no default)
Add a second modifier key to the pointer direction keys.	pointer_key_mod2	(no default)
Add a third modifier key to the pointer direction keys.	pointer_key_mod3	(no default)

Note: 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 *X0pointerkeys* 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:

Pointer Distance Functions

Movement	**Function**	**Default**
Move the pointer a number of pixels	pointer_move	10 pixels
Move the pointer using a modifier key	pointer_mod1_amt	40 pixels
Move the pointer using a modifier key	pointer_mod2_amt	1 pixel
Move the pointer using a modifier key	pointer_mod3_amt	5 pixels
Add a modifier to the distance keys	pointer_amt_mod1	no default
Add a modifier to the distance keys	pointer_amt_mod2	no default
Add a modifier to the distance keys	pointer_amt_mod3	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:

Button Operation Functions

Button Operation	**Function**	**Default Key**
Perform button 1 operations	pointer_button1_key	keypad_*
Perform button 2 operations	pointer_button2_key	keypad_/
Perform button 3 operations	pointer_button3_key	keypad_+
Perform button 4 operations	pointer_button4_key	keypad_-
Perform button 5 operations	pointer_button5_key	keypad_7

You can change the mapping of buttons on the pointer by using options in the *X*pointerkey*s 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.

Button Mapping Functions

Button Mapping	**Function**	**Default Key**
Set button 1 value	button_1_value	1
Set button 2 value	button_2_value	2
Set button 3 value	button_3_value	3
Set button 4 value	button_4_value	4
Set button 5 value	button_5_value	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:

Reset and Threshold Functions

Option	**Function**	**Default Key**
Exit the X Window System	reset	break
Add a modifier to the exit key	reset_mod1	control
Add a modifier to the exit key	reset_mod2	left_shift
Add a modifier to the exit key	reset_mod3	no default
Set the threshold for changing between screens	screen_change_amt	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_am*t 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.

Button Chording

Option	**Function**	**Default Action**
Turn button chording off or on	button_chording	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*.

Specifying a Portion of a Tablet

Option	**Function**	**Default**
Use a subset of the tablet surface as the X pointer device	tablet_subset_width tablet_subset_height tablet_subset_xorigin tablet_subset_yorigin	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*pointerkey*s 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 assignations, 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 *X0pointerkeys* file (or create an *X*pointerkeys*) to include a line assigning one of the modifier keys to *pointer_amt_mod2*. The following line in *X0pointerkeys* assigns the Left Shift key to *pointer_amt_mod2*:

```
###pointerfunction key
pointer_amt_mod2   left_shift
```

Or suppose you wanted to set up your *X0pointerkeys* 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 to the left. Pressing Left Shift 1 moves the pointer 25 pixels to the left. And pressing CTRL 1 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: 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: 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:

Valid xmodmap Expressions

To do this...	**Use this expression...**
Assign a key symbol to a keycode	*keycode <keycode> = <keysym>*
Replace a key symbol expression with another.	*keysym <keysym> = <keysym>*
Clear all keys associated with a modifier key.	*clear <modifier>*
Add a key symbol to a modifier.	*add <modifier> = <keysym>*
Remove a key symbol from a modifier.	*remove <modifier> = <keysym>*

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*.

On Hewlett-Packard keyboards, the lock modifier is set to the Caps 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", 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.

Examples

Suppose you frequently press the key at the most inopportune moments. You could remove the lock key from the lock modifier, swap it for the key, then map the 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

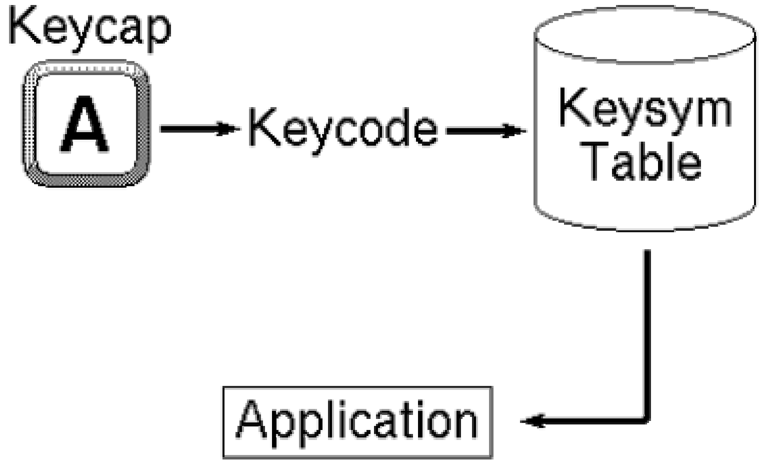
There are now two keyboards available for Hewlett-Packard workstations. In addition to the 46021 keyboard, a personal computer-style keyboard, C1429 is also available. This new keyboard is also known as the "Enhanced Vectra" keyboard.

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 12: 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.

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	Prev	Prior	Prior
Num Lock	System/User	System/User	Num_Lock
End	Select	Select	End
Page Down	Next	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
- , (on number pad)
- Tab (on number pad)

, 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.

Chapter 5 PowerShade: Enhanced 3D Rendering in Software

PowerShade is software 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.)

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: 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, 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.

3D Graphics Performance

The following information is intended to help application developers understand graphics performance on hp's Series 700 and 800 graphics devices, using three APIs: hp PEX, hp-PHIGS, and Starbase.

The information presented here does not apply to the VRX family of graphics devices (for example, PersonalVRX and TurboVRX). See the hp-UX Starbase Device Drivers Manual for more information on the VRX family. HP PEXlib is not supported on the VRX family of graphics devices.

Note: If you are not certain what graphics devices are installed 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

which is a utility installed with hp-UX. 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 tables describes how to increase application performance using the hp PEX, hp-PHIGS, and Starbase graphics APIs. These tables most accurately describe the performance factors relevant to geometry accelerator devices. Older graphics hardware (those devices without geometry accelerators) may not perform quite as well as the devices with geometry accelerators, but should show acceptable performance due to optimized paths in the software that are shared by all devices.

Maximum performance can be achieved using features listed under "Performance Optimized For:" in the following tables. 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 only specifying a single light.

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 tables were complete and accurate as of the date of publication of this document. Updates to the table, if any are necessary, will be included in the on-line release notes for the current release. See the *10.x_Rel_Notes* file in the following directories:

/opt/graphics/PEX5/newconfig/opt/graphics/PEX5
/opt/graphics/phigs/newconfig/opt/graphics/phigs
/opt/graphics/starbase/newconfig/opt/graphics/starbase

Texture Mapping

When the texture-mapping hardware is present, it is used completely, except for the following cases, which bypass the geometry accelerator hardware but continue to use other texture-mapping hardware acceleration:

- Using more than 8 lights
- Backface distinguishing
- Facet alpha

Texture-mapping hardware acceleration is completely bypassed when the following texture conditions exist:

- Multiple textures per primitive (PEX)
- Texture maps with more than 32K texels per side (Starbase, PEX)
- No texture-mapping hardware installed (Starbase, PEX)
- Prelight texturing with DECAL mode on (PEX)
- Prelight texturing with replace mode on and boundary condition absolute (external to the texture boundary is the original primitive color) (PEX)
- Starbase DECAL and the texture does not have ALPHA in it. (Starbase)

You can also resort to software texturing when you run into texture-memory space limitations or run out of texture IDs.

hp PEX Graphics Performance

Optimized vs. Normal 3D Performance

Rendering Conditions

Performance Optimized For:	**Factors Affecting Performance:**
• Light sources: o Best: up to 8 directional light sources, plus ambient o Good: a mixture of up to 8 directional and positional light sources, plus ambient • Directional eyepoint (this is the default in hp PEX unless compliance mode is enabled) • Back-face culling off, or back-face culling on with supplied geometric normals (culling usually results in a performance improvement) • View clipping on • Isotropic modeling matrix (angle-preserving) • Color approximation type PEXColorSpace • OCC primitives • Packed or stride data format	• A total of more than 8 directional and positional light sources cause a noticeable performance degradation • Picking • Any of the following ways of heavily interleaving changes: o Frequent attribute changes (e.g., frequent changes of line color) relative to the number of primitives being drawn o Frequent changes of the type of primitive (e.g., switching between Fill Areas and Polylines at a high frequency) o Frequent changes of vertex attributes (i.e., the data being supplied with each vertex) o Frequent changes of modal state, such as: ▪ Double-buffering (especially MBX or DBE) ▪ HLHSR mode or ID ▪ Light sources enabled ▪ Depth cueing state ▪ Texture mapping state o Frequent switching between two or more active Renderers o Frequent mixing of Xlib and hp PEXlib calls • Redundant or unnecessary setup (e.g. setting attributes that are not relevant to the primitives being drawn) • Deformation on • Model clipping on • HLHSR in software via PowerShade (on unaccelerated graphics devices)

Window Conditions

Performance Optimized For:	Factors Affecting Performance:
• Window unobscured or obscured only by overlay windows	• Window obscured by many other image plane windows • Backing store enabled (on supported devices)

FillArea Primitives

Performance Optimized For:	Factors Affecting Performance:
• Best: shared vertex primitives with many vertices: o *PEXTriangleStrip, PEXExtTriangleStrip, PEXOCCTriangleStrip* o *PEXSetOfFillAreaSets, PEXExtSetOfFillAreaSets, PEXOCCIndexedFillAreaSets* o *PEXOCCTriangles* o *PEXQuadrilateralMesh, PEXExtQuadrilateralMesh, PEXOCCQuadrilateralMesh* • Good: single-polygon primitives: o *PEXFillAreaSetWithData, PEXExtFillAreaSetWithData, PEXFillAreaSet, PEXOCCFillAreaSet, PEXFillArea, PEXFillAreaWithData* • Color approximation type *PEXColorSpace* • Best: edging off Fair: edging on	• More than 64 vertices in a single facet • Multiple Fill Areas in a set • Color approximation type *PEXColorRange* • Polygon offset ON (except when supported by hardware) • 2D polygons: o *PEXFillAreaSet2D, PEXFillArea2D* • These vertex data: o *PEXColorTypeIndexed* color o No vertex normals (when lighting is enabled)

Polyline Primitives

Performance Optimized For:	Factors Affecting Performance:
• Best: polylines with or without move/draw flags, many vertices per call: o *PEXPolylineSetWithData* (single set), *PEXOCCPolylines* • Polyline sets with multiple short disjoint sets: o *PEXPolylineSetWithData* (multiple sets) • Single short polylines: o *PEXPolyline, PEXOCCPolyline*	• These vertex data: o Color per vertex (*PEXColorTypeIndexed* or *PEXColorTypeRGB*) • Color approximation type *PEXColorRange*

Polymarker Primitives

Performance Optimized For:	**Factors Affecting Performance:**
• Many markers per call: o *PEXMarkers*, *PEXOCCMarkers*, *PEXhpMarkersWithData*	• Color per vertex (*PEXColorTypeIndexed*, *PEXColorTypeRGB*) • Color approximation type *PEXColorRange*

Text Primitives

Performance Optimized For:	**Factors Affecting Performance:**
• Longer strings • Best: simple text: o *PEXText*, *PEXText2D*, *PEXEncodedText*, *PEXEncodedText2D* • Fair: annotation text: o *PEXAnnotationText*, *PEXAnnotationText2D*	• Polygonal text fonts

hp-PHIGS Graphics Performance

Optimized vs. Normal 3D Performance

Performance Optimized For:	**Factors Affecting Performance:**
Rendering Conditions	
• Light Sources: o Best: up to 8 directional light sources, plus ambient o Good: a mixture of up to 8 directional and positional light sources, plus ambient • Directional eyepoint, set via: o *pescape_u230/pgse_u3* (C), *pue230/pus003* (FORTRAN) • Back face culling off, or back face culling on with supplied geometric normals • View clipping on, set via: o *pset_view_rep* (C), *psvwr* (FORTRAN) • True colour mapping in a direct colour environment • Isotropic modeling matrix (angle-preserving)	• A total of more than 8 directional and positional light sources cause a noticeable performance degradation • Picking (including direct pick and logical device picking) • Any of the following ways of heavily interleaving changes: o Frequent attribute changes (e.g., frequent changes of line colour) relative to the number of primitives being drawn o Frequent changes of the type of primitive (e.g., switching between Fill Areas and Polylines at a high frequency) o Frequent changes of vertex format (i.e., the data being supplied with each vertex) o Frequent changes of modal state, such as: ▪ HLHSR mode or ID ▪ Colour mapping method ▪ Light sources enabled ▪ Depth cueing state o Frequent switching between two or more active workstations o Frequent mixing of Xlib and hp-PHIGS calls • Redundant or unnecessary setup (e.g. setting attributes that are not relevant to the primitives being drawn) • Model clipping on • HLHSR in software via PowerShade (on unaccelerated graphics devices)

Window Conditions

Performance Optimized For:	Factors Affecting Performance:
• Window unobscured or obscured only by overlay windows	• Window obscured by other image plane windows • Backing store enabled (on supported devices)

Fill Area Set Primitives

Performance Optimized For:	Factors Affecting Performance:

• Best: shared vertex primitives with many vertices:
  o *ptri_strip3_data* (C), *ptst3d* (FORTRAN)
  o *pset_of_fill_sets3_data* (C), *psfas3* (FORTRAN)
  o *pquad_mesh3_data* (C), *pqm3d* (FORTRAN)
• Good: single-polygon primitives:
  o *pfill_area_set3* (C), *pfas3* (FORTRAN)
  o *pfill_area3* (C), *pfa3* (FORTRAN)
  o *pfill_area_set3_data* (C), *pfas3d* (FORTRAN)
• True colour mapping in a direct colour environment
• Best: edging off
• Fair: edging on
	• More than 64 vertices in a single facet
	• Multiple Fill Areas in a set
	• 2D polygons
	• Pseudo colour mapping method
	• Indexed colour environment
	• These vertex data:
	  o Indirect colour
	  o No vertex normals (when lighting is enabled)

Polyline Primitives

Performance Optimized For:	Factors Affecting Performance:

• Best: single polylines with many vertices per call:
  o *ppolyline_set3_colr* (C), *ppls3c* (FORTRAN)
  o *ppolyline3* (C), *ppl3* (FORTRAN)
  o *ppolyline* (C), *ppl* (FORTRAN)
• Polyline sets with multiple disjoint sets:
  o *pgdp_u14* (C), *pu014* (FORTRAN)
  o Plus those noted above with multiple sets
	• These vertex data:
	  o Indirect colour
	  o RGB colour
	• Pseudo colour mapping method

Polymarker Primitives

Performance Optimized For:	**Factors Affecting Performance:**
• Many markers per call: o *ppolymarker3* (C), *ppm3* (FORTRAN) o *ppolymarker* (C), *ppm* (FORTRAN)	• These vertex data: o Indirect colour o RGB colour • Pseudo colour mapping method

Text Primitives

Performance Optimized For:	**Factors Affecting Performance:**
• Longer strings • Best: simple strings: • ptext3 (C), ptx3 (FORTRAN) • ptext (C), ptx (FORTRAN) • Fair: annotation text: • panno_text_rel3 (C), patr3 (FORTRAN) panno_text_rel (C), patr (FORTRAN)	• Polygonal text fonts

Optimized vs. Normal 3D Performance

Performance Optimized For:	**Factors Affecting Performance:**
Rendering Conditions	
• Best: up to 8 directional light sources, plus ambient • Good: a mixture of up to 8 directional and positional light sources, plus ambient • Directional eyepoint set with *view_point* • Back face culling off, or back face culling on with supplied geometric normals • View clipping on • Isotropic modeling matrix (angle-preserving) • *shade_mode* set to *CMAP_FULL*	• A total of more than 8 directional and positional light sources cause a noticeable performance degradation • Picking • Any of the following ways of heavily interleaving changes: o Frequent attribute changes (e.g., frequent changes of line color) relative to the number of primitives being drawn o Frequent changes of the type of primitive (e.g., switching between polygons with polylines at a high frequency) o Frequent changes of vertex format (i.e., the data being supplied with each vertex) o Frequent changes of modal state, such as: ▪ Double-buffering ▪ Z-buffering ▪ Shade mode ▪ Light sources enabled ▪ Depth cueing state ▪ Texture mapping state o Frequent switching between two or more file descriptors o Frequent mixing of Xlib and Starbase calls • Redundant or unnecessary setup (e.g. setting attributes that are not relevant to the primitives being drawn) • Deformation on • Model clipping on • HLHSR in software via PowerShade (on unaccelerated graphics devices)

Window Conditions

Performance Optimized For:	Factors Affecting Performance:
• Window unobscured or obscured only by overlay windows	• Window obscured by many other image plane windows • Backing store enabled (on supported devices)

Polygon Primitives

Performance Optimized For:	Factors Affecting Performance:

• Best: shared vertex primitives with many vertices:
  o *triangular_strip, triangular_strip_with_data*
  o *polyhedron_with_data*
  o *quadrilateral_mesh, quadrilateral_mesh_with_data*
  o *polytriangle_with_data, polyquad_with_data*
• Good: polygon3d
• *shade_mode* set to *CMAP_FULL*
• Edging:
  o Best: edging off
  o Great: *SET_POLYGON_OFFSET* enabled, with hardware support
  o Good: *SET_POLYGON_OFFSET* enabled, with software support
  o Fair: edging on
	• More than 64 vertices in a single facet
	• 2D polygons
	• Device coordinate polygons
	• *shade_mode* set to *CMAP_NORMAL* or *CMAP_MONOTONIC*
	• Partial polygons
	• These vertex formats:
	  o Indirect color
	  o Intensity
	  o No vertex normals (when lighting is enabled)
	  o Non-unit normals

Polyline Primitives

Performance Optimized For:	Factors Affecting Performance:

• Best: polyline3d and polyline_with_data3d with or without move/draw flags, many vertices per call	• These vertex formats:
	  o RGB/vertex
	  o Indirect color
	  o Intensity
	• Individual moves and draws with move3d and draw3d
	• *shade_mode* set to *CMAP_MONOTONIC*

Polymarker Primitives

Performance Optimized For:	Factors Affecting Performance:
• Many markers per *polymarker3d* or *polymarker_with_data3d*	• These vertex formats: o RGB/vertex o Indirect color o Intensity • *shade_mode* set to *CMAP_MONOTONIC*

Text Primitives

Performance Optimized For:	Factors Affecting Performance:
• Longer strings	• Polygonal text • Spline fonts

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.

  SIGALRM
  See the "SIGALRM Details" section below for more information.

  SIGCHLD
  A multi-threaded application should not change the SIGCHLD action during the short periods when the graphics libraries are starting the Graphics Resource Manager daemon ("grmd") 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.

  SIGPIPE
  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 "SIGPIPE Details" 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: 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.

   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 t*rack(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_daemo*n).

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.

Gamma Correction

Gamma correction is used to alter hardware colormaps to compensate for the non-linearities of CRT monitors. Gamma correction can be used to improve the "ropy" or modulated appearance of antialiased lines. Gamma correction is also used to improve the appearance of shaded graphics images as well as scanned photographic images that have not already been gamma corrected.

Why Is Gamma Correction Needed?

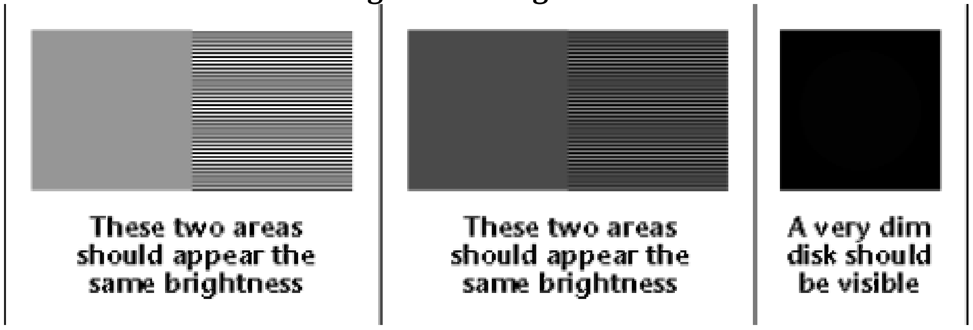
The intensity of light generated by a conventional monitor is non-linear with respect to the signal applied. The intensity output is approximately equal to the applied voltage raised to a power. This power is referred to as the **gamma** of the monitor. Stated mathematically,

$$intensity = voltage^{gamma}$$

For example, the gamma value of a properly adjusted monitor is usually between 2.35 and 2.55. Ideally, to produce an image that is 50% intensity, you would specify color values that are 50% of the maximum value. However, a monitor with a gamma of 2.4 would generate an intensity that is only 19% of the maximum intensity ($0.5^{2.4} = 0.19$).

The intensity principle is illustrated in the following test patterns. The pattern on the left has an area with exactly 50% brightness directly next to an area of horizontal stripes that alternate 100% brightness with 0% brightness. From a distance, the intensity of each area should be about the same. The middle pattern is basically the same, but just a little dimmer.

Figure 13: Brightness


These two areas should appear the same brightness
These two areas should appear the same brightness
A very dim disk should be visible

Once the brightness and contrast of your monitor are adjusted properly (see below), you can run the gamma correction tool (also described below) to see its effect on the test patterns above.

Monitor Brightness and Contrast

Often, the brightness control on a monitor is boosted to compensate for images that look dim due to a lack of gamma correction. Before applying gamma correction, the monitor brightness should be adjusted so a blank screen looks black in typical viewing conditions. Next, display a picture that is predominantly black, and adjust the brightness so that the monitor reproduces true black on the screen. The brightness should be adjusted to the threshold where it is not so far down as to "swallow" codes greater than the black code, but not so high that the picture sits on a "pedestal" of dark grey. When the critical point is reached, leave the brightness control in that position. Then set contrast to suit your preference for display intensity. How to run the Gamma Correction Tool

---

Note: You do not need to have superuser privileges to use the gamma correction tool.

To use the gamma correction tool, execute the following command:

/opt/graphics/common/bin/gamma

If you get the error "*Can't open display*", set the *DISPLAY* environment variable by executing this command:

export DISPLAY=:0.0

How to Install the Gamma Icon Into your CDE Front Panel

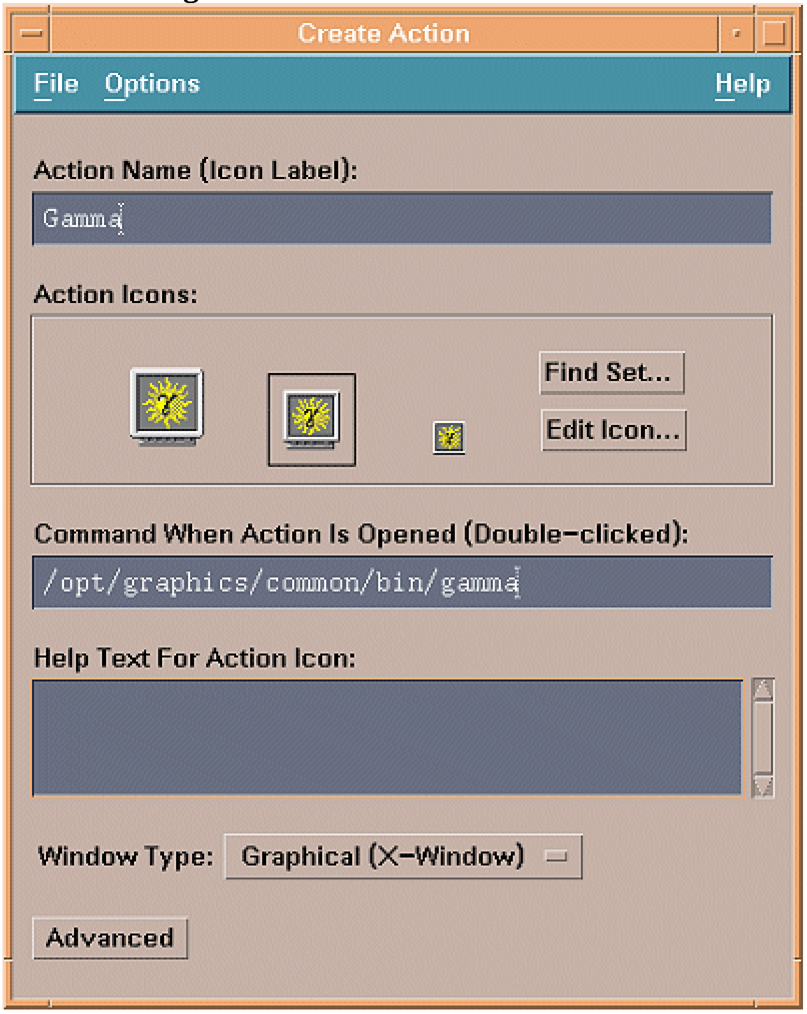
In a terminal window, go to your home directory and follow these steps:

1. Execute this command:
   cp /opt/graphics/common/icons/Gamma. $HOME/.dt/icons/*
2. Select the Applications Icon on the CDE Front Panel.
3. Double click the "Desktop_Apps" button.
4. Double click the "Create Action" button.

Use the following steps to fill in the "Create Action" form.

1. Enter Gamma in the "Action Name" window.
2. Enter /opt/graphics/common/bin/gamma in the "Command When Action is Opened" window.
3. Click "Find Set..." and double click on the .dt/icons directory under your home directory.
4. Select the "Gamma Icon" so your form appears as shown below:

Figure 14: Create Action Form



Action Name (Icon Label): Gamma

Action Icons:

Command When Action Is Opened (Double-clicked): /opt/graphics/common/bin/gamma

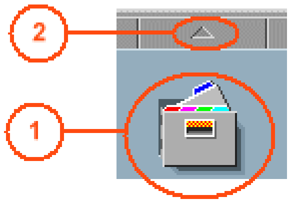
Help Text For Action Icon:

Window Type: Graphical (X-Window)

Once you have filled in the "Create Actions" form, select "Save" from the File menu and follow these steps:

1. Click the Home Folder Icon on the CDE Front Panel, and click the up-arrow above the "Home Folder Icon" to display the sub-menu.

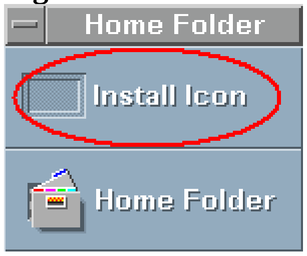
Figure 15: Home folder icon



1. Home Folder Icon
2. Arrow for displaying the “Home Folder” sub-menu

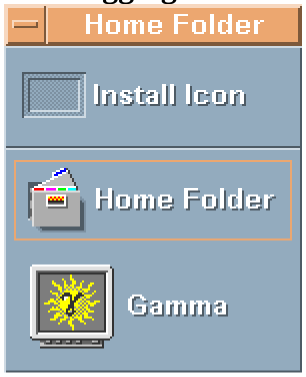
2. Drag the Gamma Icon from File Manager onto "Install Icon".

Figure 16: Install Icon



Your "Home Folder" will should look similar to Figure 17.

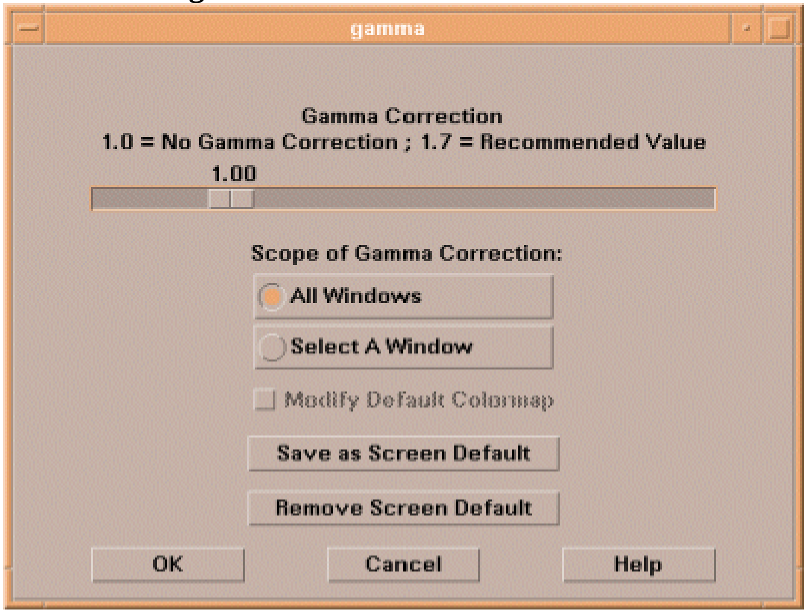
Figure 17: Home Folder after dragging the Gamma Icon to the Install Icon



If you would prefer to have the "Gamma Icon" in the Main Panel, double-click the right mouse button.

Using the Gamma Correction Tool

Figure 18: Gamma Correction Tool



Gamma Value Slider

The gamma value represents a power, so a value of 1.0 is the same as no gamma correction. Although the actual measured gamma value of most monitors is over 2.0, a gamma value of 1.7 is recommended since most applications have historically compensated for a lack of gamma correction by simply making the entire image brighter.

Scope of Gamma Correction

This area of the "Gamma correction" window allows you to select all windows to receive gamma correction or only a selected group of windows. Once you have made your selection, you can then save it as the default setting.

Selecting All Windows

To select all windows for gamma correction, single click on the "All Windows" button. Selecting this button applies gamma correction to the colormaps for all windows and will cause images, graphics, and user interfaces to all use the same specified gamma value.

Selecting Individual Windows

When you press "Select A Window," the cursor changes to a crosshair. Next, move the crosshair into the window you wish to modify and click the left mouse button. You will be able to control the gamma value for the colormap associated with the selected window.

If the colormap used by the selected window is also used by other windows, their appearance will change along with the actual window you selected.

If the only colormap used by the selected window is the default colormap, then you will need to check the "Modify Default Colormap" box as well. Modifying the default colormap will change the appearance of many windows and user-interface elements.

Save as Screen Default

When you select the "Save as Screen Default" button, the current gamma value will be saved in the appropriate X screens file (X0screens for screen 0, X1screens for screen 1, etc.). When you quit CDE and X windows, the value you save in the X screens file will be used the next time X is started.

Since the X screens file is a system resource, your system administrator determines who has permission to modify that file.

Remove Screen Default

If you select the "Remove Screen Default" button, the default gamma value saved in the X screens file will be removed and the gamma value for all windows will be reset to 1.0.

This option also requires appropriate permissions to modify the X screens file.

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). As of hp-UX 10.20, hp CDE is the default user environment, and although hp VUE is still available with hp-UX 10.20, it is not the default. 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

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.

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.

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.

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.

Finally, most X programs accept a command line option of "*-display displayname*" 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 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: 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.

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)*

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.

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 mixtures 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*".

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.

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: “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.

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.

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.

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.

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 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.

-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.

-*xnllanguagelanguage[_territory][.codeset]*

This option specifies the language, territory, and codeset for use in resolving resource and other filenames.

-*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.

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 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.

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.

Programs based on the X Tookit Intrinsics 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 for application-specific resources. For example, application default resources are usually kept in */usr/lib/X11/app-defaults*. See the X Toolkit Intrinsics - 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. 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.

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.

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:

```
$ xrdb -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 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.

```
$  xrdb $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 'rgbi:.8/.8/.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 | xwud
$  xterm -geometry 80x66-0-0 -name myxterm $*
```

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 Intrinsics encounter 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 the setting the *StringConversionsWarning* resource.

To force the X Toolkit Intrinsics 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

bdftopcf(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 Intrinsics - C Language Interface.

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).

Appendix B: X Server Reference Page

Name

X: X Window System server

Synopsis

:displaynumber [-option] <ttyname>

Description

"X" is the generic name for the window system server. It is started by the *dtlogin* 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 file name for the socket is */var/spool/sockets/X11/** where "*" is the display number.

Shared Memory IPC

If the Shared Memory Transport (*SMT*) option is enabled in the Xserver, this will be the default connection that the X Library will use to connect to an X server on the same machine if the *DISPLAY* environment variable is set to "*local*:*" or ":*" where "*" is the number of the display. Currently, *SMT* is not enabled by default. Please see */etc/X11/X0screens* and/or */usr/lib/X11/Xserver/info/screens/hp* for more information about SMT.

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.

Options

The following options can be given on the command line to the X server.

-a <number>
Sets pointer acceleration (i.e., the ratio of how much is reported to how much the user actually moved the pointer).

-audit <level>
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 4 enables messages from the *SECURITY* extension, including generation and revocation of authorizations and violations of the security policy. Level 0 turns off the audit trail. Audit lines are sent as standard-error output.

*-auth <authorization-fil*e>
Specifies a file which contains a collection of authorization records used to authenticate access.

bc
Disables certain kinds of error checking, for bug compatibility with previous releases (e.g., to work around bugs in R2 and R3 xterms and toolkits). Deprecated.

-bs
Disables backing-store support on all screens.

-c
Turns off key-click.

c <volume>
Sets key-click volume (allowable range: 0-100).

-co <filename>
Sets name of RGB color database.

-core
Causes the server to generate a core dump on fatal errors.

-dpi <resolution>
Sets the resolution of the screen, in dots per inch. To be used when the server cannot determine the screen size from the hardware.

-f <volume>
Sets beep (bell) volume (allowable range: 0-100).

-fc <cursorFont>
Sets default cursor font.

-fn <font>
Sets the default font.

-fp <fontPath>
Sets the search path for fonts. This path is a comma-separated list of directories which the server searches for font databases.

-help
Prints a usage message.

-I
Causes all remaining command-line arguments to be ignored.

-logo

Turns on the X Window System logo display in the screen-saver. There is currently no way to change this from a client. You also need to specify *-v* to enable the logo to appear.

nologo

Turns off the X Window System logo display in the screen-saver. There is currently no way to change this from a client.

-p <minutes>

Sets screen-saver pattern cycle time in minutes.

-pn

Allows X server to run even if one or more communications mechanisms fails to initialize.

-pn

Permits the server to continue running if it fails to establish all of its well-known sockets, but establishes at least one.

-r

Turns off keyboard auto-repeat.

r

Turns on keyboard auto-repeat.

-s <minutes>

Sets screen-saver timeout time in minutes.

-sp <filename>

Causes the server to attempt to read and interpret filename as a security policy file with the format described in the Security File Format section below. The file is read at server startup and reread at each server reset. The default file is */etc/X11/SecurityPolicy*.

-su

Disables *save-under* support on all screens.

-t <number>

Sets pointer acceleration threshold in pixels (i.e., after how many pixels pointer acceleration should take effect).

-terminate

Causes the server to terminate at server reset, instead of continuing to run.

-to <seconds>

Sets default connection timeout in seconds.

-tst

Disables all testing extensions (e.g., *XTEST*, *XTestExtension1*).

ttyxx

Ignored; for servers started the ancient way (from init).

-terminage

Causes server to terminate when all clients disconnect.

v

Sets video-on screen-saver preference. A window that changes regularly will be used to save the screen.

-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.

Security File Format

The syntax of the security policy file is as follows. Notation: "*" means zero or more occurrences of the preceding element, and "+" means one or more occurrences. To interpret xxx/yyy, ignore the text after the /; it is used to distinguish between instances of xxx in the next section.

<policy file>	::= <version line> <other line> *
<version line>	::= <string/v> 'n'
<other line>	::= <comment> | <access rule> | <site policy> | <blank line>
<comment >	::= # <not newline>* 'n'
<blank line>	::= <space> 'n'
<site policy >	::= <sitepolicy string/sp> 'n'
<access rule>	::= property <property/ar> <window perms> 'n'
<property>	::= <string>
<window>	::= any | <root> | <required property>
<required property>	::= <property/rp> | <property with value>
<property with value>	::= <property/rpv> = <string/rv>
<perms>	::= [<operation> | <action> | <space>]*
<operation>	::= r | w | d
<action>	::= a | i | e
<string>	::= <dbl quoted string> | <single quoted string> | <unquoted string>
<dbl quoted string>	::= <space> " <not dqoute>* " <space>
<single quoted string>	::= <space> ' <not squote>* ' <space>
<unquoted string>	::= <space> <not space>+ <space>
<space>	::= [' ' | 't']*
Character sets:	
<not newline>	::= any character except 'n'
<not dqoute>	::= any character except "
<not squote>	::= any character except '
<not space>	::= any character except those in <space>

The semantics associated with the above syntax are as follows.

<version line>, the first line in the file, specifies the file format version. If the server does not recognize the version <string/v>, it ignores the rest of the file. The version string for the file format described here is "version-1".
Once past the <version line>, lines that do not match the above syntax are ignored.

<comment lines> are ignored.

<sitepolicy> lines are currently ignored. They are intended to specify the site policies used by the *XC-QUERY-SECURITY-1* authorization method.

<access rule> lines specify how the server should react to untrusted client requests that affect the X Window property named *<property/ar>*. The rest of this section describes the interpretation of an access rule.

For an *<access rule>* to apply to a given instance of *<property/ar>*, *<property/ar>* must be on a window that is in the set of windows specified by <window>. If *<window>* is any, the rule applies to <property/ar> on any window. If *<window>* is root, the rule applies to *<property/ar>* only on root windows.

If *<window>* is required property, the following apply. If required property is a *<property/rp>*, the rule applies when the *<window>* also has that *<property/rp>*, regardless of its value. If required property is a *<property with value>*, *<property/rpv >* must also have the value specified by *<string/rv>*. In this case, the property must have type *STRING* and format 8, and should contain one or more null-terminated strings. If any of the strings match *<string/rv>*, the rule applies.

The definition of string matching is simple case-sensitive string comparison with one elaboration: the occurence of the character '*' in *<string/rv>* is a wildcard meaning "any string." A *<string/rv>* can contain multiple wildcards anywhere in the string. For example, "x*" matches strings that begin with x, "*x" matches strings that end with x, "*x*" matches strings containing x, and "x*y*" matches strings that start with x and subsequently contain y.

There may be multiple *<access rule>* lines for a given *<property/ar>*. The rules are tested in the order that they appear in the file. The first rule that applies is used.

<perms> specify operations that untrusted clients may attempt, and the actions that the server should take in response to those operations.

<operation> can be *r* (read), *w* (write), or *d* (delete). The following table shows how X Protocol property requests map to these operations in The Open Group server implementation.

GetProperty	*r, or r and d if delete=True*
ChangeProperty	*w*
RotateProperties	*r and w*
DeleteProperty	*d*
ListProperties	*none, untrus*ted clients can always list all properties

<action> can be "a" (allow), "i" (ignore), or "e" (error). "Allow" means execute the request as if it had been issued by a trusted client. "Ignore" means treat the request as a no-op. In the case of *GetProperty*, ignore means return an empty property value if the property exists, regardless of its actual value. "Error" means do not execute the request and return a BadAtom error with the atom set to the property name. Error is the default action for all properties, including those not listed in the security policy file.

An <*action*> applies to all <*operations*> that follow it, until the next <*action*> is encountered. Thus, "*irwad*" means "ignore read and write, allow delete."

GetProperty and *RotateProperties* may do multiple operations (r and d, or r and w). If different actions apply to the operations, the most severe action is applied to the whole request; there is no partial request execution. The severity ordering is:

allow < ignore < error

Thus, if the <*perms*> for a property are ired (ignore read, error delete), and an untrusted client attempts *GetProperty* on that property with *delete=True*, an error is returned, but the property value is not. Similarly, if any of the properties in a *RotateProperties* do not allow both read and write, an error is returned without changing any property values.

Here is an example security policy file.

```
version-1

# Allow reading of application resources, but not writing.
property RESOURCE_MANAGER       root    ar iw
property SCREEN_RESOURCES       root    ar iw

# Ignore attempts to use cut buffers.  Giving errors causes apps to crash,
# and allowing access may give away too much information.
property CUT_BUFFER0    root    irw
property CUT_BUFFER1    root    irw
property CUT_BUFFER2    root    irw
property CUT_BUFFER3    root    irw
property CUT_BUFFER4    root    irw
property CUT_BUFFER5    root    irw
property CUT_BUFFER6    root    irw
property CUT_BUFFER7    root    irw

# If you are using Motif, you may want these.
property _MOTIF_DEFAULT_BINDINGS        root    ar iw
property _MOTIF_DRAG_WINDOW     root    ar iw
property _MOTIF_DRAG_TARGETS    any     ar iw
property _MOTIF_DRAG_ATOMS      any     ar iw
property _MOTIF_DRAG_ATOM_PAIRS any     ar iw

# The next two rules let xwininfo -tree work when untrusted.
property WM_NAME        any     ar

# Allow read of WM_CLASS, but only for windows with WM_NAME.
# This might be more restrictive than necessary, but demonstrates
# the <required property> facility, and is also an attempt to
# say "top level windows only."
property WM_CLASS       WM_NAME ar

# These next three let xlsclients work untrusted.  Think carefully
# before including these; giving away the client machine name and command
# may be exposing too much.
property WM_STATE       WM_NAME ar
property WM_CLIENT_MACHINE      WM_NAME ar
property WM_COMMAND     WM_NAME ar

# To let untrusted clients use the standard colormaps created by
# xstdcmap, include these lines.
property RGB_DEFAULT_MAP        root    ar
property RGB_BEST_MAP   root    ar
property RGB_RED_MAP    root    ar
property RGB_GREEN_MAP  root    ar
property RGB_BLUE_MAP   root    ar
property RGB_GRAY_MAP   root    ar

# To let untrusted clients use the color management database created
# by xcmsdb, include these lines.
property XDCCC_LINEAR_RGB_CORRECTION    root    ar
property XDCCC_LINEAR_RGB_MATRICES      root    ar
property XDCCC_GRAY_SCREENWHITEPOINT    root    ar
property XDCCC_GRAY_CORRECTION  root    ar

# To let untrusted clients use the overlay visuals that many vendors
# support, include this line.
property SERVER_OVERLAY_VISUALS root    ar

# property names and explicit specification of error conditions
property "property with spaces" 'property with "'       aw er ed

# Allow deletion of Woo-Hoo if window also has property OhBoy with value
# ending in "son".  Reads and writes will cause an error.
property Woo-Hoo        OhBoy = "*son"  ad
```

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 PARENB CS7 # B9600 SANE PARENB CS7 ISTRIP IXANY       TAB3
00#X login: #Xwindow
```

There should not be a getty running against the display whenever X is run from *xinit*.

Granting Access

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
```

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.

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 CDE, errors are logged in the file */var/dt/Xerrors*.

Files

/etc/inittab	Script for the init process
/etc/gettydefs	Speed and terminal settings used by getty
/etc/X.hosts*	Initial access control list
/usr/lib/X11/fonts	Top level font directory
/usr/lib/X11/rgb.txt	Color database
/usr/lib/X11/rgb.pag	Color database
/usr/lib/X11/rgb.dir	Color database
*/usr/spool/sockets/X11/**	IPC mechanism socket
/var/dt/Xerrors	Error log file
*/etc/X11/X*devices*	Input devices used by the server. This file contains many example configurations.
*/etc/X11/X*screens*	Screens used by the server. This file contains many example configurations.
*/etc/X11/X*pointerkeys*	Keyboard pointer device file. This file contains many example configurations.
/etc/X11/Xhpkeymaps	Key device database used by the X server.
/etc/X11/SecurityPolicy	Default Security Policy file used by the X server.

Notes

The option syntax is inconsistent between 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 1996, 1998 The Open Group
Copyright 1984-1992 Massachusetts Institute of Technology
Copyright 1992 Hewlett Packard Company

See *X(1)* for a full statement of rights and permissions.

Origin

MIT Distribution

See Also

dtlogin(1),
bdftopcf(1),
fs(1),
getty(1M),
gettydefs(4),
gwindstop(1),
hpterm(1),
init(1M),
inittab(4),
mkfontdir(1),
rgb(1),
stmkdirs(1),
x11start(1),
xauth(1)
clock(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)

Appendix C: Low BandWidth X Proxy

Name

lbxproxy: Low BandWidth X proxy

Synopsis

lbxproxy [:<display>] [<option>]

Description

Applications that would like to take advantage of the Low Bandwidth extension to X (LBX) must make their connections to an *lbxproxy*. These applications don't need to know anything about LBX, they simply connect to the *lbxproxy* as if were a regular server. The *lbxproxy* accepts client connections, multiplexes them over a single connection to the X server, and performs various optimizations on the X protocol to make it faster over low bandwidth and/or high latency connections.

With regard to authentication/authorization, lbxproxy simply passes along to the server the credentials presented by the client. Since X clients will connect to *lbxproxy*, it is important that the user's .*Xauthority* file contain entries with valid keys associated with the network ID of the proxy. *lbxproxy* does not get involved with how these entries are added to the .*Xauthority* file. The user is responsible for setting it up.

The *lbxproxy* program has various options, all of which are optional.

If :<display> is specified, the proxy will use the given display port when listening for connections. The display port is an offset from port 6000, identical to the way in which regular X display connections are specified. If no port is specified on the command line option, *lbxproxy* will default to port 63. If the port number that the proxy tries to listen on is in use, the proxy will attempt to use another port number. If the proxy is not using the Proxy Manager and the default port number cannot be used, the port number that is used will be written to *stderr*.

The other command line options that can be specified are:

-*help*

Prints a brief help message about the command line options.

-*display <dpy>*

Specifies the address of the X server supporting the LBX extension. If this option is not specified, the display is obtained by the *DISPLAY* environment variable.

-*motion <count>*

A limited number of pointer motion events are allowed to be in flight between the server and the proxy at any given time. The maximimum number of motion events that can be in flight is set with this option; the default is 8.

-*maxservers <number >*

The default behavior of *lbxproxy* is to manage a single server. However, *lbxproxy* can manage more than one server. The default maximum number of servers is 20. The number of servers can be overridden by setting the environment variable *LBXPROXY_MAXSERVERS* to the desired number. The order of precedence from highest to lowest: command line, environment variable, default number.

-*[terminate|reset]*

The default behavior of *lbxproxy* is to continue running as usual when its last client exits. The -*terminate* option will cause *lbxproxy* to exit when the last client exits. The -*reset* option will cause *lbxproxy* to reset itself when the last client exits. Resetting causes *lbxproxy* to clean up its state and reconnect to the server.

-*reconnect*

The default behavior of *lbxproxy* is to exit when its connection to the server is broken. The -*reconnect* option will cause lbxproxy to just reset instead (see -*reset* above) and attempt to reconnect to the server.

-*I*

Causes all remaining arguments to be ignored.

-*nolbx*

Disables all LBX optimizations.

-*nocomp*

Disables stream compression.

-*nodelta*

Disables delta request substitutions.

-*notags*

Disables usage of tags.

-nogfx

Disables reencoding of graphics requests (not including image-related requests).

-noimage

Disables image compression.

-nosquish

Disables squishing of X events.

-nointernsc

Disables short circuiting of InternAtom requests.

-noatomsfile

Disables reading of the atoms control file. See the section on "Atom Control" for more details.

-atomsfile <file>

Overrides the default AtomControl file. See the section on "Atom Control" for more details.

-nowinattr

Disables *GetWindowAttributes*/*GetGeometry* grouping into one round trip.

-nograbcmap

Disables colormap grabbing.

-norgbfile

Disables color name to RGB resolution in proxy.

-rgbfile <path>

Specifies an alternate RGB database for color name to RGB resolution.

-tagcachesize <size>

Set the size of the proxy's tag cache (in bytes).

*-zlevel <leve>*l

Set the Zlib compression level (used for stream compression):

6: default
1: worst compression, fastest
9: best compression, slowest

-compstats

Report stream compression statistics every time the proxy resets or receives a *SIGHUP* signal.

-nozeropad

Don't zero out unused pad bytes in X requests, replies, and events.

-cheaterrors

Allows cheating on X protocol for the sake of improved performance. The X protocol guarantees that any replies, events or errors generated by a previous request will be sent before those of a later request. This puts substantial restrictions on when lbxproxy can short circuit a request. The -*cheaterrors* option allows lbxproxy to violate X protocol rules with respect to errors. Use at your own risk.

-cheatevents

The -*cheatevents* option allows *lbxproxy* to violate X protocol rules with respect to events as well as errors. Use at your own risk.

Atom Control

At startup, *lbxproxy* "pre-interns" a configurable list of atoms. This allows *lbxproxy* to intern a group of atoms in a single round trip and immediately store the results in its cache.

While running, *lbxproxy* uses heuristics to decide when to delay sending window property data to the server. The heuristics depend on the size of the data, the name of the property, and whether a window manager is running through the same *lbxproxy*.

Atom control is specified in the */etc/X11/lbxproxy/AtomControl* file, set up during installation of *lbxproxy*, with command-line overrides.

The file is a simple text file. There are three forms of lines: comments, length control, and name control. Lines starting with a "!" are treated as comments. A line of the form

z <length>

specifies the minimum length in bytes before property data will be delayed. A line of the form

<options> <atomname>

controls the given atom, where <options> is any combination of the following characters: "*i*" means the atom should be pre-interned; "*n*" means data for properties with this name should never be delayed; and "*w*" means data for properties with this name should be delayed only if a window manager is also running through the same *lbxproxy*.

Notes

When the authorization protocol *XDM-AUTHORIZATION-1* is used:

A client must be on the same host as *lbxproxy* for the client to be authorized to connect to the server.

If a client is not on the same host as *lbxproxy*, the client will not be authorized to connect to the server.

Origin

The Open Group

See Also

proxymngr(1),
xfindproxy(1)

Appendix D: RX Netscape Navigator Plug-in

Name

libxrx: RX Netscape Navigator Plugin

Description

The RX Plugin may be used with Netscape Navigator (3.0 or later) to interpret documents in the RX MIME type format and start remote applications.

The RX Plugin reads an RX document, from which it gets the list of services the application wants to use. Based on this information, the RX Plugin sets the various requested services, including creating authorization keys. It then passes the relevant data, such as the X display name, to the application through an HTTP GET request of the associated CGI script. The Web server then executes the CGI script to start the application. The client runs on the web server host connected to your X server. In addition when the RX document is used within the *<embed>* tag (a Netscape extension to HTML), the RX Plugin uses the *XC-APPGROUP* extension, to cause the remote application to be embedded within the browser page from which it was launched.

Installation

To install the RX Plugin so that Netscape Navigator can use it, copy the file named */usr/lib/X11R6/libxrx.6.3* to either */usr/local/lib/netscape/plugins* or *$HOME/.netscape/plugins*.

If you have configured Netscape Navigator to use the RX helper program (xrx), you must reconfigure it. Generally you simply need to remove or comment out the line you may have previously added in your mailcap file to use the RX helper program. Otherwise the plugin will not be enabled. (The usual comment character for mailcap is #.)

If you are already running Netscape Navigator, you need to exit and restart it after copying the plugin library so the new plugin will be found. Once this is done you can check that Navigator has successfully loaded the plugin by checking the About Plugins page from the Help menu. This should show something like:

RX Plugin

File name: /usr/local/lib/netscape/plugins/libxrx.6.3

X Remote Activation Plugin

Mime Type	*Description*	*Suffixes*	*Enabled*
application/x-rx	*X Remote Activation Plugin*	*xrx*	*Yes*

Once correctly configured, Netscape Navigator will activate the RX Plugin whenever you retrieve any document of the MIME type application/x-rx.

Resources

The RX Plugin looks for resources associated with the *widget netscape*.Navigator (*class Netscape.TopLevelShell*) and understands the following resource names and classes:

xrxFastWebServers (class XrxFastWebServers)

The web servers for which LBX should not be used. Its value is a comma-separated list of mask/value pairs to be used to filter web servers, based on their address. The mask part specifies which segments of the address are to be considered and the value part specifies what the result should match. For instance, the following list:

255.255.255.0/198.112.45.0, 255.255.255.0/198.112.46.0

matches the address sets: *198.112.45.** and *198.112.46.**. More precisely, the test is:

(address & mask) == value.

xrxTrustedWebServers (*class XrxTrustedWebServers*)

The web servers from which remote applications should be run as trusted clients. The default is to run remote applications as untrusted clients. The resource value is a list of address mask/value pairs, as previously described.

Environment

If the RX document requests *X-UI-LBX* service and the default X server does not advertise the LBX extension, the RX Plugin will look for the environment variable *XREALDISPLAY* to get a second address for your X server and look for the LBX extension there. When running your browser through lbxproxy you will need to set *XREALDISPLAY* to the actual address of your server if you wish remote applications to be able to use LBX across the Internet.

If the RX document requests XPRINT service, RX Plugin looks for the variable *XPRINTER* to get the printer name and X Print server address to use. If the server address is not specified as part of XPRINTER, RX Plugin uses the first one specified through the variable XPSERVERLIST when it is set. When it is not RX Plugin then tries to use the video server as the print server. If the printer name is not specified via *XPRINTER*, RX Plugin looks for it in the variables *PDPRINTER*, then *LPDEST*, and finally *PRINTER*.

Notes

When an authorization key is created for a remote application to use the X Print service, the RX Plugin has to create the key with an infinite timeout since nobody knows when the application will actually connect to the X Print server. It then revokes the key when its instance is destroyed (that is when you go to another page). However, if the plugin does not get destroyed properly, which happens when Netscape Navigator dies unexpectedly, the print authorization key will never get revoked.

Origin

The Open Group

See Also

xrx(1),
lbxproxy(1),
proxymngr(1)

Appendix E: Proxy Manager Service

Name

proxymngr: Proxy Manager Service

Synopsis

proxymngr [-config <filename>] [-timeout <seconds>] [-retries <#>-verbose]

Description

The proxy manager (*proxymngr*) is responsible for resolving requests from *xfindproxy* (and other similar clients), starting new proxies when appropriate, and keeping track of all of the available proxy services. The proxy manager strives to reuse existing proxies whenever possible.

There are two types of proxies that the proxy manager deals with: **managed** and **unmanaged** proxies. A **managed** proxy is a proxy that is started "on demand" by the proxy manager. An **unmanaged** proxy, on the other hand, is started either at system boot time, or manually by a system administrator. The proxy manager is made aware of its existence, but no attempt is made by the proxy manager to start unmanaged proxies.

The command line options that can be specified to *proxymngr* are:

-*config*
: Used to override the default *proxymngr config* file. See below for more details about the config file.

-*timeout*
: Sets the number of seconds between attempts made by the proxy manager to find an unmanaged proxy. The default is 10.

-*retries*
: Sets the maximum number of retries made by the proxy manager to find an unmanaged proxy. The default is 3.

-*verbose*
: Causes various debugging and tracing records to be displayed as requests are received and proxies are started.

Proxy Manager Config File

The proxy manager maintains a local configuration file describing the proxy services available. This configuration file is installed in */etc/X11/proxymngr/pmconfig* during the installation of *proxymngr*. The location of the configuration file can be overwritten using the -config command line option.

Aside from lines starting with an exclamation point for comments, each line of the configuration file describes either an unmanaged or managed proxy service.

For unmanaged proxies, the format is:

<service-name> unmanaged <proxy-address>

<service-name> is the name of the unmanaged proxy service, and must not contain any spaces; for example, LBX. service-name is case insenstive.

<proxy-address> is the network address of the unmanaged proxy. The format of the address is specific to the *<service-name>*. For example, for the LBX service, the *<proxy-address>* might be lbx.x.org:100.

If there is more than one entry in the config file with the same unmanaged *<service-name>*, the proxy manager will try to use the proxies in the order presented in the config file.

For managed proxies, the format is:

<service-name> managed <command-to-start-proxy>

<service-name> is the name of the managed proxy service, and must not contain any spaces, for example LBX. service-name is case insensitive.

<command-to-start-proxy> is the command executed by the proxy manager to start a new instance of the proxy. If *<command-to-start-proxy>* contains spaces, the complete command should be surrounded by single quotes. If desired, *<command-to-start-proxy>* can be used to start a proxy on a remote machine. The specifics of the remote execution method used to do this is not specified here.

Example

Here is a sample configuration file:

```
!  proxy manager config file
!
!  Each line has the format:
!     managed
!        or
!     unmanaged
!
lbx managed /usr/bin/X11/lbxproxy -display dispName:0.0 :1
!
```

Proxy Manager Details

When the proxy manager gets a request from *xfindproxy* (or another similar client), its course of action will depend on the *<service-name>* in question.

For a managed proxy service, the proxy manager will find out if any of the already running proxies for this service can handle a new request. If not, the proxy manager will attempt to start up a new instance of the proxy (using the *<command-to-start-proxy>* found in the config file). If that fails, an error will be returned to the caller.

For an unmanaged proxy service, the proxy manager will look in the config file to find all unmanaged proxies for this service. If there is more than one entry in the config file with the same unmanaged *<service-name>*, the proxy manager will try to use the proxies in the order presented in the config file. If none of the unmanged proxies can satisfy the request, the proxy manager will timeout for a configurable amount of time (specified by *-timeout* or default of 10) and reattempt to find an unmanaged proxy willing to satisfy the request. The number of retries can be specified by the *-retries* argument, or a default of 3 will be used. If the retries fail, the proxy manager has no choice but to return an error to the caller (since the proxy manager cannot start unmanaged proxy services).

Notes

The proxy manager listen port should be configurable.

-*timeout* and -*retries* are not implemented in proxymngr.

proxymngr does not utilize the "*options*" and "*host*" fields in the proxy management protocol *GetProxyAddr* request.

Origin

The Open Group

See Also

xfindproxy(1),
lbxproxy(1)

Appendix F: Locate Proxy Services

Name

xfindproxy: Locate Proxy Services

Synopsis

xfindproxy -manager <managerAddr > -name <serviceName> -server <serverAddr> [-auth] [-host <hostAddr>] [-options <opts>]

Description

xfindproxy is a program used to locate available proxy services. It utilizes the Proxy Management Protocol to communicate with a proxy manager. The proxy manager keeps track of all available proxy services, starts new proxies when necessary, and makes sure that proxies are shared whenever possible.

The -*manager* argument is required, and it specifies the network address of the proxy manager. The format of the address is a standard ICE network ID (for example, "tcp/info.x.org:6500").

The -*name* argument is required, and it specifies the name of the desired proxy service (for example, "LBX"). The name is case-insensitive.

The -*server* argument is also required, and it specifies the address of the target server. The format of the address is specific to the proxy service specified with the -*name* argument. For example, for a proxy service of "LBX", the address would be an X display address (e.g, "info.x.org:0").

The -*auth* argument is optional. If specified, xfindproxy will read two lines from standard input. The first line is an authorization/authentication name, and the second line is the authorization/authentication data in hex format (the same format used by xauth). xfindproxy will pass this auth data to the proxy, and in most cases, will be used by the proxy to authorize/authenticate itself to the target server.

The -*host* argument is optional. If xfindproxy starts a new proxy service, it will pass the host specified. The proxy may choose to restrict all connections to this host. In the event that *xfindproxy* locates an already existing proxy, the host will be passed, but the semantics of how the proxy uses this host are undefined.

The -*options* argument is optional. If *xfindproxy* starts a new proxy service, it will pass any options specified. The semantics of the options are specific to each proxy server and are not defined here. In the event that *xfindproxy* locates an already existing proxy, the options will be passed, but the semantics of how the proxy uses these options are undefined.

If *xfindproxy* is successful in obtaining a proxy address, it will print it to stdout. The format of the proxy address is specific to the proxy service being used. For example, for a proxy service of "LBX", the proxy address would be the X display address of the proxy (e.g, "info.x.org:63").

If *xfindproxy* is unsuccessful in obtaining a proxy address, it will print an error to *stderr*.

Origin

The Open Group

See Also

proxymngr(1),
lbxproxy(1)

Appendix G: RX Helper Program

Name

xrx: RX Helper Program

Synopsis

xrx [-<toolkitoption> . . .]<filename>

Description

The helper program may be used with any Web browser to interpret documents in the RX MIME type format and start remote applications.

xrx reads in the RX document specified by its <*filename*>, from which it gets the list of services the application wants to use. Based on this information, *xrx* sets the various requested services, including creating authorization keys if your X server supports the SECURITY extension. It then passes the relevant data, such as the X display name, to the application through an HTTP GET request of the associated CGI script. The Web server then executes the CGI script to start the application. The client runs on the web server host connected to your X server.

Installation

You need to configure your web browser to use *xrx* for RX documents. Generally the following line in your *$HOME/.mailcap* is enough:

```
application/x-rx; xrx %s
```

However, you may need to refer to your web browser's documentation for exact instructions on configuring helper applications.

Once correctly configured, your browser will activate the helper program whenever you retrieve any document of the *MIME type application/x-rx*.

Options

The *xrx* helper program accepts all of the standard X Toolkit command line options such as:

-xrm <resourcestring>

This option specifies a resource string to be used. There may be several instances of this option on the command line.

Resources

The application class name of the <*xrx*> program is *Xrx* and it understands the following application resource names and classes:

xrxFastWebServers (class XrxFastWebServers)

The web servers for which LBX should not be used. Its value is a comma-separated list of mask/value pairs to be used to filter web servers, based on their address. The mask part specifies which segments of the address are to be considered and the value part specifies what the result should match. For instance, the following list:

255.255.255.0/198.112.45.0, 255.255.255.0/198.112.46.0

matches the address sets: *198.112.45.** and *198.112.46.**. More precisely, the test is:

(address & mask) == value.

xrxTrustedWebServers (class XrxTrustedWebServers)

The web servers from which remote applications should be run as trusted clients. The default is to run remote applications as untrusted clients. The resource value is a list of address mask/value pairs, as previously described.

Environment

The <*xrx*> helper program uses the standard X environment variables such as *DISPLAY* to get the default X server host and display number. If the RX document requests *X-UI-LBX* service and the default X server does not advertise the LBX extension, <*xrx*> will look for the environment variable *XREALDISPLAY* to get a second address for your X server and look for the *LBX* extension there. When running your browser through *lbxproxy* you will need to set *XREALDISPLAY* to the actual address of your server if you wish remote applications to be able to use *LBX* across the Internet.

If the RX document requests *XPRINT* service, <*xrx*> looks for the variable *XPRINTER* to get the printer name and X Print server address to use. If the server address is not specified as part of *XPRINTER*, <*xrx*> uses the first one specified through the variable *XPSERVERLIST* when it is set. When it is not *xrx* then tries to use the video server as the print server. If the printer name is not specified via *XPRINTER*, <*xrx*> looks for it in the variables *PDPRINTER*, then *LPDEST*, and finally *PRINTER*,

Notes

When an authorization key is created for a remote application to use the X Print service, the helper program has to create the key with an infinite timeout since nobody knows when the application will actually connect to the X Print server. Therefore, in this case, the helper program stays around to revoke the key when the application goes away (that is when its video key expires). However, if the helper program dies unexpectedly the print authorization key will never get revoked.

Origin

The Open Group

See Also

libxrx(1),
lbxproxy(1),
proxymngr(1)