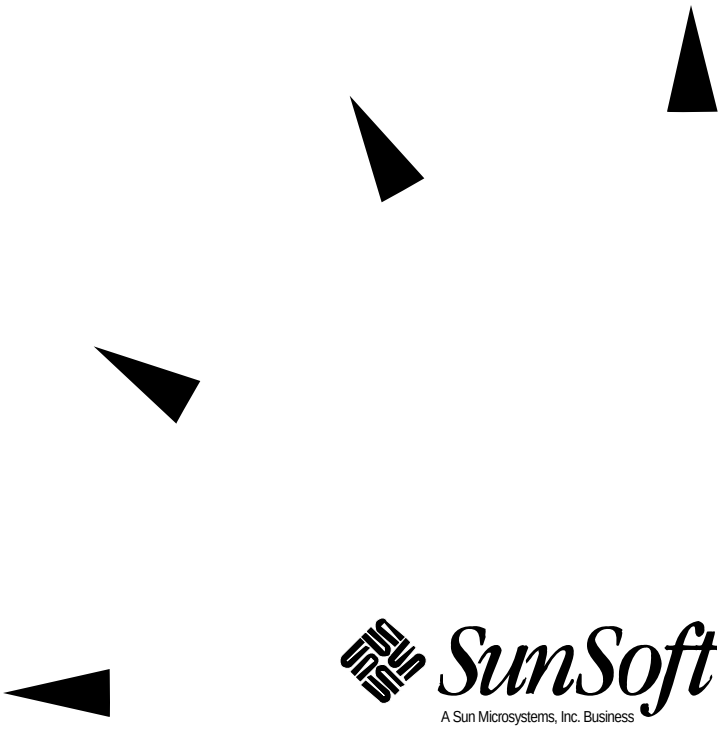
Solaris X Window System Developer's Guide



2550 Garcia Avenue
Mountain View, CA 94043
U.S.A.

SunSoft
A Sun Microsystems, Inc. Business

Please
Recycle

Adobe PostScript

Contents

Preface xiii

New Features xvii

1. Introduction to the Solaris X Server 1

About the Solaris X Server 1

X11R5 Sample Server 2

DPS Extension 4

X Consortium Extensions 4

AccessX 6

Shared Memory Transport 7

Visual Overlay Windows 7

X11 Libraries 8

Applications That Run With the Solaris X Server 8

Supported X11 Applications 9

Unsupported Applications 10

OpenWindows Directory Structure 10

Notes on X11 Programming 13
Compose Key Support 13
NumLock Key Support 13
Color Name Database 14
Color Recommendations. 14
Further Reading 15
2. DPS Features and Enhancements. 17
About DPS 17
How DPS Works. 18
DPS Font Enhancements in the Solaris Server 20
DPS Libraries 20
Adobe NX Agent Support 20
DPS Security Issues 21
System File Access. 21
Secure Context Creation 21
When DPS Encounters Internal Errors 23
How To Access Information From Adobe. 23
DPS Compositing Operators 24
Operator Descriptions. 26
Implementation Notes and Limitations 31
3. Visuals on the Solaris X Server. 35
About Visuals 35
Default Visual. 36
Visuals on Multi-Depth Devices 37

Hints for Windows Programming With Visuals 37
Gamma-Corrected Visuals 38
Finding a Linear Visual 39
Visual Selection Alternatives 41
4. Font Support 43
Font Support in the Solaris X Server 43
Available Font Formats 44
Associated Files 45
Outline and Bitmap Fonts 46
Replacing Outline Fonts With Bitmap Fonts 47
Using F3 Fonts in DPS 47
Locating Fonts 48
Changing the Default Font Path in X11 48
Changing the Resource Path in DPS 50
Adding New Fonts 51
Adding Bitmap Fonts 52
Adding Outline Fonts 53
Using OPEN LOOK Fonts on X Terminals 56
5. Transparent Overlay Windows 57
What are Transparent Overlay Windows? 57
Basic Characteristics of Overlay Windows 59
Overlay Window Paint Type 59
Overlay Window Viewability 60
Rendering Transparent Paint 60

More on Overlay Window Characteristics 60
Overlay Window Background 61
Overlay Window Border 62
Overlay Window Backing Store 62
Overlay Window Gravity 63
Overlay Colormap 63
Input Distribution Model 63
Print Capture 64
Choosing Visuals for Overlay / Underlay Windows 65
Example Program 66
Overview of the Solaris Transparent Overlay Window API . . . 67
Creating Overlay Windows 68
Setting the Paint Type of a Graphics Context 70
Setting the Background State of an Overlay Window 71
Rendering to an Overlay Window 71
Querying the Characteristics of an Overlay Window 72
Determining Whether a Window is an Overlay Window . . 72
Determining the Paint Type of a Graphics Context 72
Pixel Transfer Routines 73
Filling an Area Using the Source Area Paint Type 73
Copying an Area and Its Paint Type 75
Retrieving Overlay Color Information 79
Using Existing Xlib Pixel Transfer Routines With Overlay Windows 81

Designing an Application for Portability 82
Selecting a Visual for an Overlay / Underlay Window 83
Selecting an Optimal Overlay / Underlay Visual Pair 88
6. Security Issues 93
Access Control Mechanisms 94
User-Based 94
Host-Based 94
Authorization Protocols 95
MIT-MAGIC-COOKIE-1 95
SUN-DES-1 95
Changing the Default Authorization Protocol 96
Manipulating Access to the Server 97
Client Authority File 98
Allowing Access When Using MIT-MAGIC-COOKIE-1. . . 99
Allowing Access When Using SUN-DES-1. 99
Running Clients Remotely, or Locally as Another User 100
A. Reference Display Devices 101
Solaris Reference Display Devices 101
SPARC: Supported Reference Devices 102
x86: Supported Reference Devices 104
B. Multi-Buffering Application Program Interface, Version 3.2 107
Library File 107
Header File 108
New Routines 108

New Types 108
New Constants 108
New Structures 110
MBX Functions 111
Glossary 127
Index 133

Figures

Figure 1-1 Solaris X Server 2

Figure 1-2 Solaris X Server Architecture 3

Figure 1-3 OpenWindows Directory Structure 11

Figure 2-1 DPS Extension to X 19

Figure 2-2 Compositing Operator Example Program 26

Figure 2-3 Results of Compositing Operations 29

Tables

Table 1-1	X11 Libraries	8
Table 1-2	OpenWindows Directories	11
Table 2-1	DPS Libraries	20
Table 2-2	Factors of the Compositing Equation	28
Table 4-1	Outline Font Formats	44
Table 4-2	Bitmap Font Formats	44
Table 4-3	Font File Availability	45
Table 4-4	Bitmap Font Binaries	46
Table 4-5	Font Directory Structure	48
Table 5-1	Background Values for an Overlay Window	61
Table 5-2	List of Overlay Window Routines	67
Table 5-3	XSolarisOvlCopyPaintType Source / Destination Combinations and Actions	74
Table 5-4	XSolarisOvlCopyAreaAndPaintType Source / Destination Combinations and Actions	78
Table A-1	Solaris Reference Display Devices	102

Preface

The *Solaris X Window System Developer's Guide* provides detailed information on the Solaris™ X server. The guide provides an overview of the server architecture and tells you where to look for more information.

This guide provides detailed information for software developers interested in interfacing with the Solaris X server.

Who Should Use This Book

Programming in this environment primarily involves using a toolkit and possibly interfacing with the server and its protocols. The protocols and toolkits are documented elsewhere (see "Related Books" on page xv). Read this manual if you need detailed information on the:

- Features of the Solaris X server
- Differences from and enhancements to the X Consortium sample server
- DPS imaging system
- Supported display devices
- Authorization schemes and protocols for server connections

Before You Read This Book

This manual assumes that the reader has a programming background and familiarity with, or access to, appropriate documentation for:

- Solaris 2.x
- X Window System™
- C programming language
- PostScript™
- The Display PostScript™ System (DPS)
- `olwm` window manager
- XView™ toolkit

How This Book Is Organized

Although you can read this book in sequence, it is designed for you to read only those chapters of interest. This book serves both as an overview and as a reference document.

Chapter 1, “Introduction to the Solaris X Server,” describes the architecture of the Solaris X server, the X and DPS extensions, Sun’s enhancements to the X Consortium libraries and extensions, notes on color-related issues, and a list of applications you can run with the server.

Chapter 2, “DPS Features and Enhancements” describes the DPS features specific to Solaris and includes information on compositing operators provided as an extension to standard DPS.

Chapter 3, “Visuals on the Solaris X Server,” describes visuals in the Solaris environment. It also provides hints for window programming with visuals.

Chapter 4, “Font Support,” describes the set of fonts provided and how to use and add fonts.

Chapter 5, “Transparent Overlay Windows,” describes the Solaris application programming interface (API) for transparent overlay windows.

Chapter 6, “Security Issues,” describes the security features of the Solaris environment.

Appendix A, “Reference Display Devices,” describes the graphics devices provided as reference devices with the Solaris environment.

Appendix B, "Multi-Buffering Application Program Interface, Version 3.2," describes the C language API to the multi-buffering extension.

Related Books

For information on how to write applications in the Solaris environment, consult the following manuals:

- *Desktop Integration Guide*
- *ToolTalk User's Guide*
- *OpenWindows Desktop Reference Manual*
- *X Window System Reference Manual*
- *XView Developer's Notes*
- *OLIT QuickStart Programmer's Guide*
- *OLIT Reference Manual*

The following X-related manuals are available through SunExpress or your local bookstore. Contact your SunSoft representative for information on ordering any of these books.

- *XView Reference Manual*, O'Reilly & Associates
- *XView Programming Manual*, O'Reilly & Associates
- *Xlib Reference Manual*, O'Reilly & Associates
- *Xlib Programming Manual*, O'Reilly & Associates
- *X Protocol Reference Manual*, O'Reilly & Associates
- *Programmer's Supplement for Release 5*, O'Reilly & Associates
- *X Toolkit Intrinsics Reference Manual*, O'Reilly & Associates
- *X Window System, Third Edition*, Digital Press
- *The X Window System Server, X Version 11, Release 5*, Digital Press

The following PostScript and DPS-related manuals are available through SunExpress or your local bookstore. Contact your SunSoft representative for information on ordering.

- *PostScript Language Reference Manual, Second Edition*, Adobe® Systems Incorporated
- *PostScript Language Tutorial and Cookbook*, Adobe Systems Incorporated
- *Programming the Display PostScript System with X*, Adobe Systems Incorporated
- *PostScript Language Program Design*, Adobe Systems Incorporated
- *Adobe Type I Font Format*, Adobe Systems Incorporated

What Typographic Changes and Symbols Mean

Table P-1 describes the type changes and symbols used in this book.

Table P-1 Typographic Conventions

Typeface or Symbol	Meaning	Example
`AaBbCc123`	The names of commands, files, and directories; on-screen computer output	Edit your `.login` file. Use `ls -a` to list all files. `system% You have mail.`
`AaBbCc123`	What you type, contrasted with on-screen computer output	`system% su` `password:`
AaBbCc123	PostScript programming language commands	Use the `currentpath` operator.
AaBbCc123	Command-line placeholder: replace with a real name or value	To delete a file, type `rm` *filename*.
AaBbCc123	Book titles, new words or terms, or words to be emphasized	Read Chapter 6 in *User's Guide*. These are called *class* options. You *must* be root to do this.
Code samples are included in boxes and may display the following:		
%	UNIX C shell prompt	`system%`
$	UNIX Bourne shell prompt	`system$`
#	Superuser prompt, either shell	`system#`

New Features

The following new features are provided in this release of the Solaris X server.

Shared Memory Transport Extension

The Solaris X server now includes the Sun extension SUN_SME, Sun's implementation of a shared memory transport mechanism. This extension provides the capability of sending client requests to the server via shared memory.

MT-Safe DGA

The Direct Graphics Access extension to the Solaris server is now MT-safe. This means that the extension can take advantage of multi-threaded processing.

DPS Toolkit for Motif

The Display PostScript Toolkit for Motif is available in Solaris at this release. The DPS Motif Toolkit is a set of Motif widgets that assist in common programming tasks. These tasks include font selection, color selection, and scrolling and zooming within a DPS application. For information on this toolkit, refer to *Programming the Display PostScript System with X* from Adobe Systems Incorporated.

NumLock Key Support

In this release, the OpenWindows version of Xlib supports NumLock Key processing through calls to `XLookupString`. For information, see Chapter 1, “Introduction to the Solaris X Server”.

DPS Compositing Operators

The OpenStep™ compositing extension to the Display PostScript system has been added to the Solaris server at this release. Compositing enables separately rendered images to be combined into a final image. For information on this feature, see Chapter 2, “DPS Features and Enhancements”.

New Features in the DPS Extension

The following new features are provided in the DPS extension to the X11 server at this release:

- CID fonts – Native support for Adobe’s CID font format has been provided. The CID font format was designed for large character set fonts. Using CID fonts allows for improved performance over Adobe’s Type 0 (composite) counterparts. See Adobe Technical Note #5092 (CID-Keyed Font File Format Overview) for more information.
- XATM 3.0 – XATM is the interface that allows X11 clients to access fonts such as Type 1 fonts, which are rendered by the Display PostScript extension. The new XATM calls into the PostScript interpreter directly to do its font rasterization. This means that X11 clients can now use any font that XDPS can handle, including Type 3 fonts, Type 0 (composite) fonts, and CID fonts. Previous releases of XATM only allowed X11 clients to use Type 1 fonts.

Introduction to the Solaris X Server 1

This chapter provides information on the Solaris X server. The Solaris X server implements the X Window System client-server model for the Solaris product. The chapter includes information on the following topics:

- Features of the Solaris X server, including supported extensions from the X Consortium and the Display PostScript extension
- Supported and unsupported X11 applications
- OpenWindows™ directory structure

About the Solaris X Server

The Solaris X server, `XSun`, is composed of the X Consortium's X11R5 sample server with the Display PostScript (DPS) imaging system extension, additional X Consortium X extensions, and Sun added value. The Solaris X server is the foundation for the OpenWindows environment and underlies the OPEN LOOK® desktop. The server handles communication between client applications, the display hardware, and input devices. By default, the Solaris X server runs with the OPEN LOOK window manager (`olwm`), but any X Window System manager that is ICCCM (Inter-Client Communication Conventions Manual) compliant runs with the server. Software developers can write applications for the Solaris environment using the Xlib library or a variety of toolkits, including the Motif toolkit, the Xt toolkit, or the OpenWindows toolkits XView and the OPEN LOOK Intrinsics Toolkit (OLIT).

Figure 1-1 illustrates the relationship between the Solaris X server, several desktop client applications, the display, and input devices.

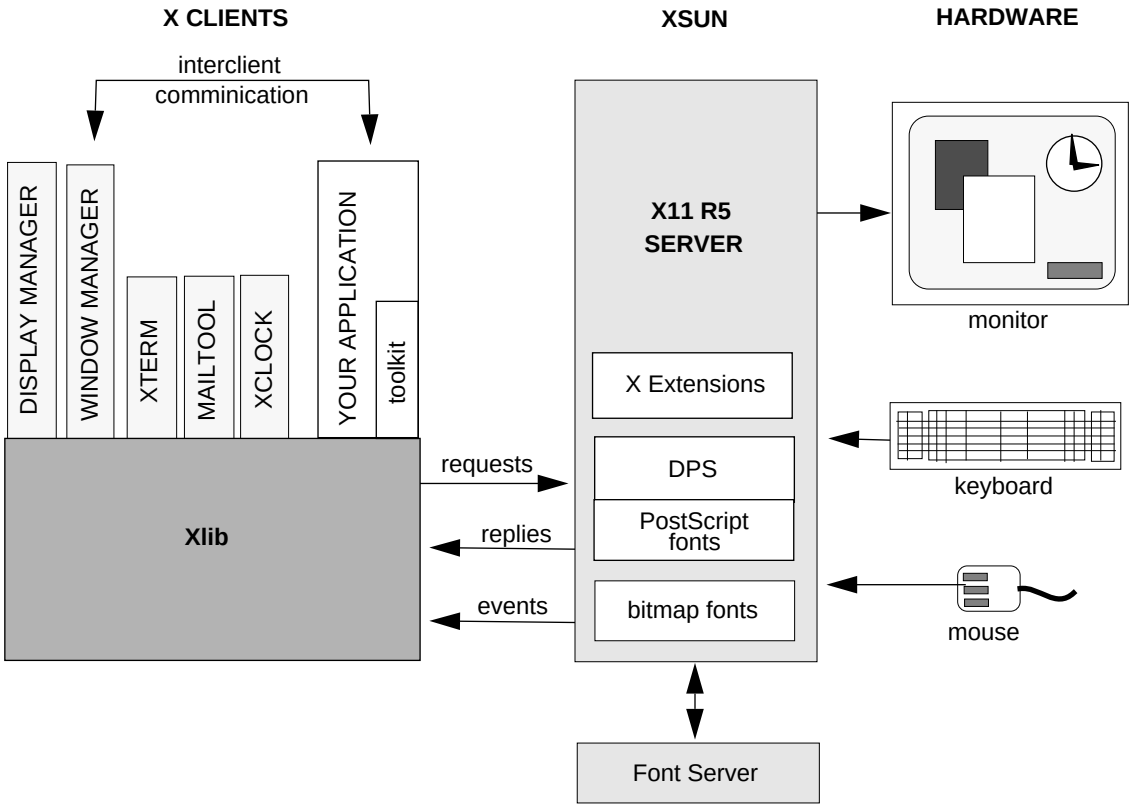


Figure 1-1 Solaris X Server

X11R5 Sample Server

An important component of the Solaris X server is the X11R5 sample server from the X Consortium. The X11R5 sample server was designed and implemented to be *portable*; it hides differences in the underlying hardware from client applications. The sample server handles all drawing, interfaces with device drivers to receive input, and manages off-screen memory, fonts, cursors, and colormaps.

The sample server contains the following parts, or *layers*:

- Device-Independent Layer (DIX) – Dispatches client requests, manages the event queue, distributes events to clients, and manages visible data structures. This layer contains functions that do not depend on graphics hardware, input devices, or the host operating system.
- Device-Dependent Layer (DDX) – Creates and manipulates pixmaps, clipping regions, colormaps, screens, fonts, and graphics contexts. In addition, the DDX layer collects events from input devices and relays them to the DIX layer. This layer contains routines that depend on graphics hardware and input devices the server must accommodate.
- Operating System Layer (OS) – Manages client connections and connection authorization schemes, and provides routines for memory allocation and deallocation. The OS layer contains functions that rely on the host operating system.
- Font Management Library – The font management library enables the server to use font files of different formats and to load fonts from the X font server. The server's font features are described in detail in Chapter 4, "Font Support."

Figure 1-2 illustrates the structure of the server. Note that throughout this document, *server* is used interchangeably with the Solaris X server, and *sample server* is used interchangeably with the X Consortium's X11R5 sample server.

Server Architecture

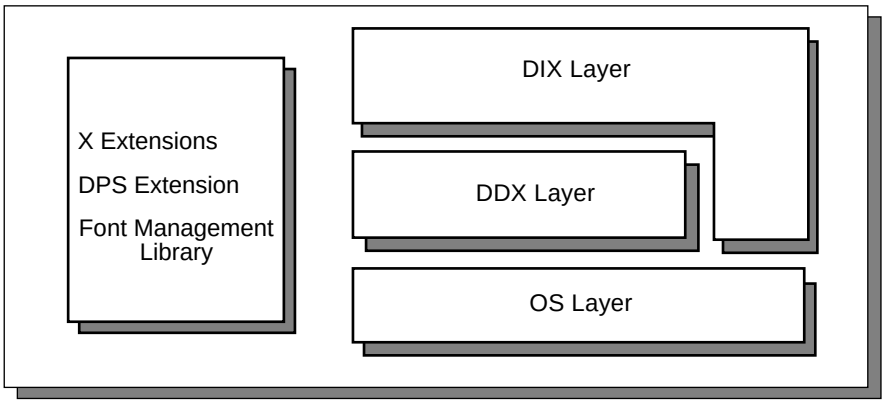


Figure 1-2 Solaris X Server Architecture

1

DPS Extension

In addition to the X11R5 sample server, the Solaris X server includes the Display PostScript system. DPS provides X applications with the PostScript imaging model and with access to the Adobe Type Library. The Display PostScript system is implemented as an extension to the X Window System as part of the client-server network architecture; the extension is sometimes referred to as *DPS/X*.[1]

In the DPS system, the PostScript interpreter is implemented as an extension to the X server, and each application is a client. The application sends PostScript language code to the server through single operator calls, and data can be returned from the server in the form of output arguments. DPS client-server communication is implemented transparently using the low-level communication protocols provided by the X Window System. For more information on the DPS system, see Chapter 2, "DPS Features and Enhancements".

X Consortium Extensions

The Solaris X server supports six X extensions as defined (or proposed) by the X Consortium. These extensions are briefly described in the sections below. The sections provide the specification name for each extension, as well as the associated file name (on `ftp.x.org`) in parentheses. For information on the standard X Extension Mechanism, see *The X Window System Server* and the *Xlib Programming Manual.*

The X Consortium X11 standards referenced in the following sections are readily available to systems on Internet. The X11 documentation resides in the `/pub/R5untarred/mit/doc/extensions` directory on the `ftp.x.org` machine. Use the file transfer protocol (`ftp`) to download files from this system. If you need help using `ftp`, refer to the `ftp(1)` man page. To determine if your system is connected to Internet, see your system administrator.

1. This section is based on Chapter 2 of *Programming the Display PostScript System with X* by Adobe Systems Incorporated (Addison-Wesley Publishing Company, Inc., 1993) and is used with the permission of the copyright holder.

X Input Extension

The X Input Extension is Sun's implementation of the X Consortium standard, *X11 Input Extension Protocol Specification* (`/xinput/protocol.ms`). This extension controls access to alternate input devices (that is, other than the keyboard and pointer). It allows client programs to select input from these devices independently of each other and independently of the core devices.

Multi-Buffering Extension

The Multi-Buffering Extension (MBX) is Sun's implementation of the X Consortium proposed standard, *Extending X for Double-Buffering, Multi-Buffering, and Stereo*. This specification is located in `/usr/openwin/share/src/extensions/mbx-spec-3.2.ps`—it is not on the `ftp.x.org` machine. This extension provides the capability of creating and displaying multiple drawable buffers for each window, and displaying a rapid succession of buffers in a window to achieve smooth animation. The stereo windows portion is *not* implemented in Sun's Multi-Buffering Extension. See Appendix B, "Multi-Buffering Application Program Interface, Version 3.2," for more information.

Caution – In future releases, Sun's MBX implementation will change when this proposed standard is approved as an X Consortium standard. Backward compatibility is not guaranteed.

Shape Extension

The Shape Extension is Sun's full implementation of the X Consortium standard, *X11 Nonrectangular Window Shape Extension* (`shape.ms`). This extension provides the capability of creating arbitrary window and border shapes within the X11 protocol.

Shared Memory Extension

The Shared Memory extension is Sun's full implementation of the X Consortium experimental Shared Memory Extension (`mit-shm.ms`). This extension provides the capability to share memory `XImages` and pixmaps by storing the actual image data in shared memory. This eliminates the need to

move data through the Xlib interprocess communication channel; thus, for large images, system performance increases. This extension is useful only if the client application runs on the same machine as the server.

XTEST Extension

The XTEST extension is Sun's full implementation of the X Consortium proposed standard, *X11 Input Synthesis Extension Proposal* (`xtest1.mm`). This extension provides the capability for a client to generate user input and to control user input actions without a user being present. This extension requires modification to the DDX layer of the server.

Miscellaneous Extension

The MIT-SUNDRY-NONSTANDARD extension was developed at MIT and does not have a standard, or specification, on the `ftp.x.org` machine. This extension handles miscellaneous erroneous protocol requests from X11R3 and earlier clients. It provides a request that turns on bug-compatibility mode so that certain erroneous requests are handled or turns off bug-compatibility mode so that an error for erroneous requests is returned. The extension also provides a request that gets the current state of the mode.

This extension can be dynamically turned on or off with `xset`, or at server startup with `openwin`. See the `xset(1)` and `openwin(1)` man pages, specifically the `-bc` option, for more information.

AccessX

The Solaris X server also supports keyboard features compliant with the American Disabilities Act (ADA). These features are available through an extension to the server, called AccessX. The AccessX extension provides the following capabilities: sticky keys, slow keys, toggle keys, mouse keys, bounce keys and repeat keys. Use the client program `accessx` to enable and disable these capabilities. The `accessx` client controls the toggle, bounce, and repeat keys and their settings. The sticky, slow, and mouse keys can be enabled using shift or other keys. For information on using AccessX, see the *Solaris User's Guide*.

Before running `accessx`, set the UIDPATH environment variable to `/usr/openwin/lib/app-defaults/accessx.uid`.

The `accessx` client is part of the `SUNWxwacx` package. To install it, you need to install the `All Cluster`.

Shared Memory Transport

The Solaris X server includes the Sun extension SUN_SME, Sun's implementation of a shared memory transport mechanism. This extension provides the capability of sending client requests to the server via shared memory. Shared memory is used for client requests only. Replies from the server and events are sent via the default transport mechanism. To enable this transport mechanism, set the `DISPLAY` environment variable to `:x.y`, where x is the display number, and y is the screen number, and set the environment variable `XSUNTRANSPORT` to `shmem`. The size of the segment can be set by setting the environment variable `XSUNSMESIZE` to the desired size in Kbytes. By default, `XSUNSMESIZE` is set to 64.

Visual Overlay Windows

The Solaris X server also includes an application programmer's interface (API) that enables transparent overlay windows. An overlay is a pixel buffer (either physical or software-simulated) into which graphics can be drawn. Applications can use overlays to display temporary imagery in a display window. For more information on the visual overlay API, see Chapter 5, "Transparent Overlay Windows".

1

X11 Libraries

Table 1-1 lists the X11 libraries. The `.so` and `.a` files that comprise these libraries are in `/usr/openwin/lib`.

Table 1-1 X11 Libraries

Library	Description	Available From the X Consortium	Sun Value Added
`libX11`	Xlib	Yes	MT safe Dynamic loading of locale Search path includes `/usr/openwin`, New keysyms
`libXau`	X Authorization library	Yes	None
`libXaw`	Athena Widget Set library	Yes	None
`libXext`	X Extensions library	Yes	Bug fixes, transparent overlays
`libXinput`	Binary compatibility library for previous input extension	No	Sun library
`libXi`	Xinput Extension library	Yes	Bug fixes Supports Solaris X extensions
`libXmu`	X Miscellaneous Utilities library	Yes	Search path includes`/usr/openwin`
`libXol`	OLIT library	No	Sun product—see the preface for a list of OLIT manuals (Available from USL)
`libXt`	Xt Intrinsics library	Yes	None
`libxview`	XView library	Yes	Sun product donated to X Consortium Bug fixes not included in X11R5 `libxview`

Applications That Run With the Solaris X Server

You can run the following kinds of applications with the Solaris X server:

- Applications written with the following toolkits:
  - OpenWindows toolkits: OLIT and XView
  - Motif toolkit
  - Xt toolkit
- Applications written for the X protocol
- Applications written for the DPS interface
- SPARC OpenWindows Version 3 X11 applications compiled under SunOS 4.x

Note – The OpenWindows Version 3 X11 applications must adhere to the system Binary Compatibility Package. See the *Binary Compatibility Guide* for more information.

- x86 Applications from Interactive Unix

Applications written with the following interfaces are *not* supported:

- TNT, NeWS, and XVPS
- SunView, SunWindows, and Pixrect

Supported X11 Applications

The Solaris X server supports the following client applications available from the X Consortium. These clients are also included as part of the Solaris environment.

- `xterm` terminal emulator
- `twm` window manager
- `xdm` display manager
- `bitmap` bitmap editor
- `xfd` font display utility
- `xauth` access control program
- `xhost` access control utility
- `xrdb` resource control program
- `xset` user preference setting program
- `xsetroot` root window appearance setting utility
- `xmodmap` keyboard control utility
- `xlsfonts` server font listing utility
- `xfontsel` font selection utility

- `xlswins` window listing utility
- `xwininfo` window information utility
- `xlsclients` client applications information utility
- `xdpyinfo` server information display utility
- `xprop` window and font properties utility

Unsupported Applications

The following are some applications and libraries, all of which are available from the X Consortium, that run on the server but are *not* distributed or supported by Sun:

- Andrew, InterViews
- The `uwm` and `wm` window managers
- The CLX Common Lisp interface
- *contrib* X Consortium clients

OpenWindows Directory Structure

The OpenWindows directory structure, which includes the Solaris X server executable and X11 core distribution libraries, is shown in Figure 1-3 on page 11. Note that `/openwin/etc` is a symbolic link to `/openwin/share/etc`, `/openwin/include` is a link to `/openwin/share/include`, and `/openwin/man` is a link to `/openwin/share/man`. The `/share` directory contains architecture-independent files.

For more information on the X11 libraries in `/openwin/lib`, see page 8.

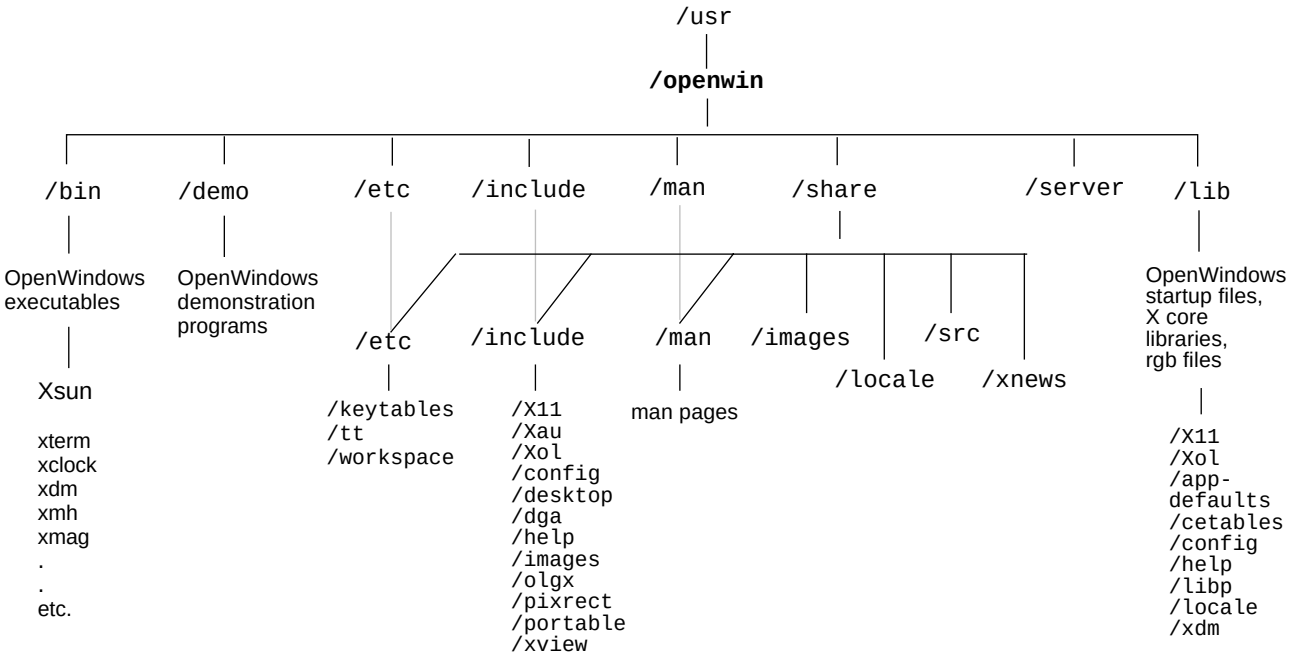


Figure 1-3 OpenWindows Directory Structure

Table 1-2 briefly describes the contents of the top level directories in the OpenWindows directory structure.

Table 1-2 OpenWindows Directories

Directory	Subdirectory	Content
`/etc`	`/keytables`	US and international keytables, and `keytable.map`
	`/tt`	ToolTalk® data files
	`/workspace`	`/patterns` (`.xbm` files and attributes)
`/include`	`/X11`	X11 header files, `/DPS`, `/Xaw`, `/Xmu`, `/bitmaps`, `/extensions`
	`/Xau`	Symbolic link to `/include/X11`
	`/Xol`	OLIT header files
	`/config`	`generic.h` header file
	`/desktop`	Classing engine header files

Table 1-2 OpenWindows Directories *(Continued)*

Directory	Subdirectory	Content
	/dga	dga.h header file
	/help	libhelp header files
	/images	Various bitmap files
	/olgx	olgx header file
	/pixrect	Pixrect header files
	/portable	c_varieties.h and portable.h header files
	/xview	XView header files
/lib	/X11	Server support files, /fonts, and DPS .upr files
	/Xol	OLIT data files
	/app-defaults	X applications default files
	/cetables	Classing Engine tables
	/config	imake files
	/help	Symbolic link to /locale/C/help
	/libp	Profiles libraries
	/locale	Locale libraries (/C, /iso_8859_1)
	/xdm	Xdm configuration files
/man	/man1, /man1m	OpenWindows command man pages
	/man3	Library man pages, for XView, OLIT, Xt, Xlib, etc.
	/man4	AnswerBook man pages
	/man5	File format man pages
	/man6	Demos man pages
	/man7	Non-command man pages
/server	Server private files for internal use only	
/share	/etc	Location of files in /etc
	/images	/PostScript, /fish, /raster
	/include	Location of files in /include
	/locale	Location of files in /lib/locale

Table 1-2 OpenWindows Directories *(Continued)*

Directory	Subdirectory	Content
	/man	Location of files in /man
	/src	/dig_samples, /extensions, /fonts, /olit, /tooltalk, /xview
	/xnews	/client

Notes on X11 Programming

Common X11 programming issues are discussed in the following sections.

Compose Key Support

The OpenWindows version of Xlib supports Compose Key processing through calls to XLookupString.

For x86 systems – On x86 keyboards, use the Control-Shift-F1 key sequence for the Compose Key functionality.

NumLock Key Support

The OpenWindows version of Xlib supports NumLock Key processing through calls to XLookupString. This change does not affect the NumLock processing that exists in XView, OLIT, Motif, or X applications.

For x86 systems – On x86 keyboards, the NumLock Key resides in the top line of the keypad section of the keyboard..

Color Name Database

The color name database provides a mapping between ASCII color names and RGB color values. This mapping increases the portability of color programs and eases programming. Note that this mapping is subjective and has no objective scientific basis.

The source of the database is `/usr/openwin/lib/X11/rgb.txt`. This file is identical to the one provided in X11R5 from the X Consortium. `rgb.txt` is compiled into the `dbm(3)` database files, `rgb.dir` and `rgb.pag`. When the server starts up, it builds an internal representation of `rgb.dir` and `rgb.pag` used to map a color name to a color value.

X11 clients use `XLookupColor` or `XAllocNamedColor` to map a color name to a color value. The color name string passed to these routines is converted to lowercase before it is looked up in the database.

Color Recommendations

This section contains recommendations for using the Solaris X server color support facilities. Use these hints to maximize portability and color sharing:

- Do not rely on the locations of black and white in the default PseudoColor colormap. Always use `XAllocColor` to allocate a pixel for rendering.

Note – Do not rely on black and white being in certain pixel locations. Future versions of the Solaris X server and the servers of other vendors may have these colors located in different positions than the current server. For maximum portability and compatibility, always write X11 clients so that they use the `XAllocColor` function to allocate desired colors for rendering.

- Do not use a visual before you have checked on all supported visual types, using `XGetVisualInfo` or `XMatchVisualInfo`. Note that `XGetVisualInfo` is the recommended function to use because it has the ability to distinguish between visuals of the same class and depth.
- To reduce colormap flashing, it is usually a good policy to try to first allocate colors from the default colormap. Only when this allocation fails should you create a private colormap.
- For more hints on writing portable X11 color clients, see “Hints for Windows Programming With Visuals” on page 37.

Further Reading

There are numerous books on all aspects of X and the X Window System. For more information on the X Window System, see page xv of the preface for a list of recommended books available through SunExpress and your local book store. For more information on the Solaris X server and the X Consortium sample server, see the following manual pages:

- `Xsun(1)` – Solaris X server
- `Xserver(1)` – the X Consortium sample server
- `openwin(1)` – OpenWindows startup command

1

DPS Features and Enhancements 2

This chapter provides information on the Display PostScript (DPS) extension to the Solaris X server. The following topics are briefly discussed:

- Overview information on the DPS system
- Solaris font enhancements to DPS
- DPS security issues
- DPS compositing operators

About DPS

The Display PostScript system displays graphical information on the computer screen with the same PostScript language imaging model that is a standard for printers and typesetters.[1] The PostScript language makes it possible for an X application to draw lines and curves with perfect precision, rotate and scale images, and manipulate type as a graphic object. In addition, X applications that use the Display PostScript system have access to the entire Adobe Type Library.

1. This section is based on Chapter 4 of *Programming the Display PostScript System with X* by Adobe Systems Incorporated (Addison-Wesley Publishing Company, Inc., 1993) and is used with the permission of the copyright holder.

Device and resolution independence are important benefits of PostScript printers and typesetters. The Display PostScript system extends these benefits to interactive displays. An application that takes advantage of the DPS system will work and appear the same on any display without modification to the application program.

How DPS Works

The DPS system has several components, including the PostScript interpreter, the Client Library, and the `pswrap` translator. The Client Library is the link between an application and the PostScript interpreter.

Each application that uses the DPS extension creates a *context*. A context can be thought of as a virtual PostScript printer that sends its output to a window or an offscreen pixmap. It has its own set of stacks, input/output facilities, and memory space. Separate contexts enable multiple applications to share the PostScript interpreter, which runs a single process in the server.

Although the DPS system supports multiple contexts for a single application, one context is usually sufficient for all drawing within an application. A single context can handle many drawing areas. There are exceptions, however, when it is preferable to use more than one context in a client. For example, a separate context might be used when importing Encapsulated PostScript (EPS) files. This simplifies error recovery if an included EPS file contains PostScript errors.

An application draws on the screen by making calls to Client Library procedures. These procedures generate PostScript language code that is sent to the PostScript interpreter for execution. In addition to the Client Library, the DPS system provides the `pswrap` translator. It takes PostScript language operators and produces a C-language procedure–called a wrap–that can then be called from an application program.

The PostScript interpreter handles the scheduling associated with executing contexts in time slices. The interpreter switches among contexts, giving multiple applications access to the interpreter. Each context has access to a private portion of PostScript virtual memory space (VM). An additional portion of VM, called *shared VM*, is shared among all contexts and holds system fonts and other shared resources. *Private VM* can hold fonts private to the context.

Figure 2-1 shows the components of DPS and their relationship to X.

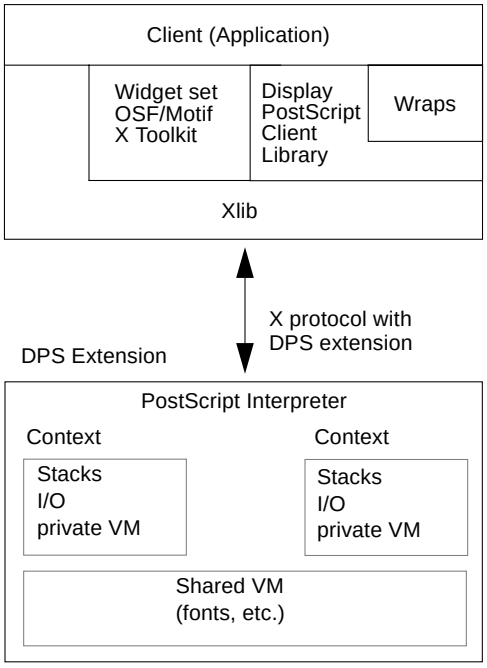


Figure 2-1 DPS Extension to X

An application interacts with the DPS system in the following manner:

1. The application creates a PostScript execution context and establishes a communication channel to the server.
2. The application sends Client Library procedures and wraps to the context and receives responses from it.
3. When the application exits, it destroys the context and closes the communications channel, freeing resources used during the session.

The structure of a context is the same across all DPS platforms. Creating and managing a context, however, can differ from one platform to another. The *Client Library Reference Manual* and *Client Library Supplement for X* contain information on contexts and the routines that manipulate them, and *Display PostScript Toolkit for X* contains utilities for Display PostScript developers.

2

DPS Font Enhancements in the Solaris Server

The Solaris X server includes the following font enhancements to the DPS system:

- Support for F3 Latin and Asian fonts
- Support for obtaining prescaled bitmap font formats from X11 font code
- Type 1 fonts (`.pfa` and `.pfb`)

See Chapter 4, “Font Support,” for more information.

DPS Libraries

Table 2-1 lists the DPS libraries. The `.so` and `.a` files that comprise these libraries are located in the `/usr/openwin/lib` and `/usr/openwin/lib/libp` directories. For information on these libraries, see *Programming the Display PostScript System with X* and *PostScript Language Reference Manual*.

Table 2-1 DPS Libraries

Library	Description
`libdps`	DPS Client library
`libdpstk`	DPS Toolkit library
`libpsres`	PostScript Language Resource Location library
`libdpstkXm`	DPS Motif Toolkit library

Adobe NX Agent Support

The context creation routines (`XDPSCreateSimpleContext` and `XDPSCreateContext`) in `libdps` attempt to contact the DPS NX agent if they are unable to connect to the DPS/X extension. The NX client must be started manually, usually during the boot or X startup process.

The Adobe DPS NX agent, which is available from Adobe Systems Inc., is a separate process from the X server and the DPS/X client. When connected to the DPS NX agent, the client’s DPS calls are intercepted and converted into standard X Protocol requests. Thus, a DPS client can run on an X server that does not natively support the DPS extension.

DPS Security Issues

The Solaris environment provides, and in some cases exceeds, the X Consortium's X11R5 sample server security levels. In particular, DPS programmers should be aware of two DPS-specific security features: PostScript file operators' inability to access system files, and secure context creation. These features are described below.

System File Access

The PostScript language provides file operations that allow users to access system devices such as disk files. This presents a serious security problem. In the Solaris environment, you cannot—by default—use PostScript file operators to open or otherwise access a system file.

For applications, the client rather than the server should perform necessary file operations. Thus, the client does not need all the same access privileges that the server needs. If you want PostScript file operators to access system files, start the server with the `-dpsfileops` option (see the `Xsun(1)` man page). If you attempt to access system files without specifying `-dpsfileops`, you will get a PostScript `undefinedfilename` error. This issue is particularly important in the CDE or `xdm` environment, as the server process is owned by a super-user.

Secure Context Creation

DPS contexts normally have access to global data. This allows a context to look into the activities of another context. For example, one context could intercept a document that another context is imaging. This section describes how to create secure contexts in the Solaris environment.

Section 7.1.1, "Creating Contexts," in the *PostScript Language Reference Manual, Second Edition* describes three ways that contexts can share VM:

1. "Local and global VM are completely private to the context." This capability is new with Level 2, and a context created this way is called a *secure context*.
2. "Local VM is private to the context, but global VM is shared with some other context." This is the normal situation for contexts created with `XDPSCreateContext` and `XDPSCreateSimpleContext`.

3. “Local and global VM are shared with some other context.” This is the situation for contexts created with `XDPSCreateContext` and `XDPSCreateSimpleContext` when the `space` parameter is not `NULL`.

To create a secure context, use `XDPSCreateSecureContext` as shown below:

```
XDPSCreateSecureContext DPSContext XDPSCreateSecureContext(dpy,
        drawable, gc, x, y, eventmask, grayramp, ccube, actual,
        textProc, errorProc, space)
    Display *dpy;
    Drawable drawable;
    GC gc;
    int x;
    int y;
    unsigned int eventmask;
    XStandardColormap *grayramp;
    XStandardColormap *ccube;
    int actual;
    DPSTextProc textProc;
    DPSErrorProc errorProc;
    DPSSpace space;
```

All parameters have the identical meaning to those in `XDPSCreateContext`, but the context being created has its own private global VM. If the `space` parameter is not `NULL`, it must identify a space created with a secure context. A space created with a secure context cannot be used for the creation of a nonsecure context. Specifying a nonsecure space with a secure context or a secure space with a nonsecure context generates an access error.

When DPS Encounters Internal Errors

DPS conducts consistency checks during execution. In the rare event that it encounters internal errors, DPS applications will not be able to connect to the server. If this happens, you must restart the Solaris environment. If a client tries to connect to a server with the DPS extension in this state, the following error message sometimes appears:

```
XError: 130
Request Major code 129 (Adobe-DPS_Extension)
```

How To Access Information From Adobe

The following information is readily available from Adobe's public access file server: source code examples, Adobe Metric Font (AMF) files, documentation, PostScript printer description (PPP) files, and press releases. You can obtain this information if you have access to the Internet or UUCP electronic mail.

If you have access to the Internet, use the file transfer protocol (`ftp`) program to download files from the `ftp.mv.us.adobe.com` machine. Read the `README.first` file for information on the archived files. For details on obtaining information from Adobe by electronic mail, see the "Public Access File Server" section in the preface of *Programming the Display PostScript System with X*.

2

DPS Compositing Operators

Warning – The operators defined in this section are extensions to the Display PostScript language. They are not part of the standard DPS and thus are not available in all DPS implementations. An application that depends on these operators is not portable and cannot display on servers that do not support these operators.

Compositing is an OpenStep™ extension to the Display PostScript system. Compositing enables separately rendered images to be combined into a final image. It encompasses a wide range of imaging capabilities:

- It provides a means for simply copying an image as is from one place to another with PostScript.
- It allows two images to be added together so that both appear in the composite superimposed on each other.
- It defines a number of operations that take advantage of transparency in one or both images that are combined. When the images are composited, the transparency of one image can let parts of the other image show through.

Compositing can be used for copying within the same window, as during scrolling, or for taking an image rendered in one drawable and transferring it to another. In OpenStep applications, images are often stored in pixmaps and composited into windows as they are needed.

When images are partially transparent, they can be composited so that the transparent sections of one image determine what the viewer sees of the other. Each compositing operation uses transparency in a different way. In a typical operation, one image provides a background or foreground for the other. When parts of an image are transparent, it can be composited over an opaque background, which will show though transparent “holes” in the image on top. In other operations, transparent sections of one image can be used to “erase” matching sections of the images it is composited with. In most operations, the composite is calculated from the transparency of both images.

Compositing with transparency can achieve a variety of interesting visual effects. A partially transparent, uniformly gray area can be used like a pale wash to darken the image it is composited with. Patches of partially

transparent gray can add shadows to another image. Repeated compositing while slowly altering the transparency of two images can dissolve one into another. Or an animated figure can be composited over a fixed background.

Before images can be composited, they must be rendered. To take advantage of transparency when compositing, at least one of the images needs to be rendered with transparent paint.

The following PostScript program fragment shows the use of the compositing operators. The program creates two simple images and composites them. The first image, the destination, is a 0.8 gray triangle on a white backround; the second, the source, is a 0.6 gray triangle on a transparent background.

```
% Create the Destination triangle
    0.8 setgray
    100 100 moveto
    100 0 rlineto
    0 -100 rlineto
    fill

% Make the background of the source transparent
    0 setalpha
    0 0 100 100 rectfill

% Draw the Source triangle
    1 setalpha
    0.6 setgray
    0 0 moveto
    0 100 rlineto
    100 0 rlineto
    fill

% Compute the result
    0 0 100 100 null 100 0 Sover composite
```

The eighth operand to the composite operator, Sover, defines how the source and destination pixels are combined. In the example, the opaque parts of the source image are placed over the destination image. The resulting image looks like Figure 2-2 on page 26.

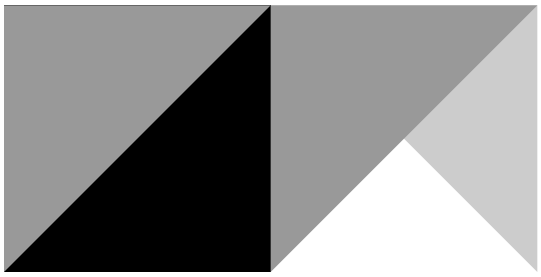


Figure 2-2 Compositing Operator Example Program

Operator Descriptions

This section describes the new DPS operators. The information is provided in the format used in the PostScript manuals *PostScript Language Reference Manual* and *Programming the Display PostScript System with X*.

setalpha *coverage* **setalpha**

Sets the *coverage* parameter in the current graphics state to *coverage*. *coverage* should be a number between 0 and 1, with 0 corresponding to transparent, 1 corresponding to opaque, and intermediate values corresponding to partial coverage. The default value is 1. This establishes how much background shows through for purposes of compositing. If the coverage value is less than 0, the coverage parameter is set to 0. If the value is greater than 1, the coverage parameter is set to 1.

The coverage value affects the color painted by PostScript marking operations. The current color is pre-multiplied by the alpha value before rendering. This multiplication occurs after the current color has been transformed to RGB space.

Errors **stackunderflow**, **typecheck**

See also **composite**, **currentalpha**

currentalpha - **currentalpha** *coverage*

Returns the coverage parameter of the current graphics state.

Errors **None**

See also **composite, setalpha**

composite *srcx srcy width height srcgstate destx desty op* **composite**

Performs the compositing operation specified by *op* between pairs of pixels in two images, a source and a destination. The source pixels are in the drawable referred to by the *srcgstate* graphics state, and the destination pixels are in the drawable specified by the current graphics state. If *srcgstate* is `NULL`, the current graphics state is assumed.

The rectangle specified by *srcx*, *srcy*, *width*, and *height* defines the source image. The outline of the rectangle may cross pixel boundaries due to fractional coordinates, scaling, or rotated axes. The pixels included in the source are all those that the outline of the rectangle encloses or enters.

The destination image has the same size, shape, and orientation as the source; *destx* and *desty* give destination's location image compared to the source. Even if the two graphic states have different orientations, the images will not; composite will not rotate images.

Both images are clipped to the frame rectangles of the respective drawables. The destination image is further clipped to the clipping path of the current graphics state. The result of a composite operation replaces the destination image.

op specifies the compositing operation. The color of each destination image pixel (alpha value) after the operation, dst' (dstA'), is given by:

```
dst' = src * Fs(srcA, dstA, op) + dst * Fd(srcA, dstA, op)
dstA' = srcA * Fs(srcA, dstA, op) + dstA * Fs(srcA, dstA, op)
```

where `src` and `srcA` are the source color and alpha values, `dst` and `dstA` are the destination color and alpha values, and `Fs` and `Fd` are the functions given in Table 2-2 on page 28.

The choices for the composite *op* are given in Table 2-2. See Figure 2-3 on page 29 for the result of each operation.

Errors **rangecheck, stackunderflow, typecheck**

See also **compositerect, setalpha, setgray, sethsbcolor, setrgbcolor**

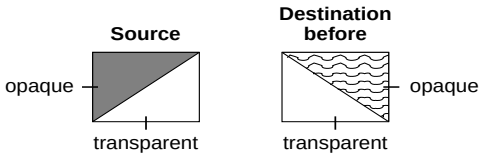
Table 2-2 Factors of the Compositing Equation

Op	Fs	Fd
Clear	0	0
Copy	1	0
Sover	1	1 - srcA
Sin	dstA	0
Sout	1 - dstA	0
Satop	dstA	1 - srcA
Dover	1 - dstA	1
Din	0	srcA
Dout	0	1 - srcA
Datop	1 - dstA	srcA
Xor	1 - dstA	1 - srcA
PlusD[1]	N / A	N / A
PlusL[2]	1	1

1. PlusD does not follow the general equation. The equation is dst'=(1-dst)+(1-src). If the result is less than 0 (black), then the result is 0.

2. For PlusL, the addition asturates. That is, if (src+dst) > white), the result is white.

Figure 2-3 on page 29 shows the result of the compositing operations.



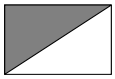
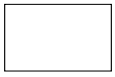
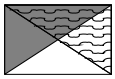
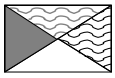
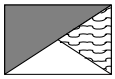
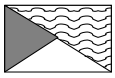
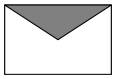
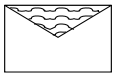
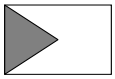
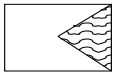
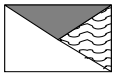
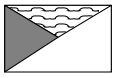
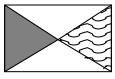
Operation	Destination after	
Copy		Source image.
Clear		Transparent.
PlusD		Sum of source and destination images, with color values approaching 0 as a limit.
PlusL		Sum of source and destination images, with color values approaching 1 as a limit.
Sover		Source image wherever source image is opaque, and destination image elsewhere.
Dover		Destination image wherever destination image is opaque, and source image elsewhere.
Sin		Source image wherever both images are opaque, and transparent elsewhere.
Din		Destination image wherever both images are opaque, and transparent elsewhere.
Sout		Source image wherever source image is opaque but destination image is transparent, and transparent elsewhere.
Dout		Destination image wherever destination image is opaque but source image is transparent, and transparent elsewhere.
Satop		Source image wherever both images are opaque, destination image wherever destination image is opaque but source image is transparent, and transparent elsewhere.
Datop		Destination image wherever both images are opaque, source image wherever source image is opaque but destination image is transparent, and transparent elsewhere.
Xor		Source image wherever source image is opaque but destination image is transparent, destination image wherever destination image is opaque but source image is transparent, and transparent elsewhere.

Figure 2-3 Results of Compositing Operations

compositerect *destx desty width height op* **compositerect -**

In general, this operator is the same as the composite operator except that there is no real source image. The destination is in the current graphics state; *destx*, *desty*, *width*, and *height* describe the destination image in that graphics state's current coordinate system. The effect on the destination is as if there were a source image filled with the color and coverage specified by the graphics state's current color and coverage parameters. *op* has the same meaning as the *op* operand of the composite operator; however, one additional operation, **Highlight**, is allowed.

Highlight turns every white pixel in the destination rectangle to light gray and every light gray pixel to white, regardless of the pixel's coverage value. Light gray is defined as 2/3. Repeating the same operation reverses the effect. (On monochrome displays, Highlight inverts each pixel so that white becomes black, black becomes white.)

Note – The **Highlight** operation doesn't change the value of a pixel's coverage component. To ensure that the pixel's color and coverage combination remains valid, **Highlight** operations should be temporary and should be reversed before any further compositing.

For **compositerect**, the pixels included in the destination are those that the outline of the specified rectangle encloses or enters. The destination image is clipped to the frame rectangle and clipping path of the window in the current graphics state.

Errors **rangecheck, stackunderflow, typecheck**

See also **composite, setalpha, setgray, sethsbcolor, setrbgcolor**

dissolve *srcx srcy width height srcgstate destx desty delta* **dissolve -**

The effect of this operation is a blending of a source and a destination image. The first seven arguments choose source and destination pixels as they do for composite. The exact fraction of the blend is specified by *delta*, which is a floating-point number between 0.0 and 1.0. The resulting image is:

```
delta * source + (1-delta) * destination
```

If *srcgstate* is null, the current graphics state is assumed.

Errors **stackunderflow, typecheck**

See also **composite**

The values of the composite *op* are available for applications in the PostScript **systemdict.** The definitions are as follows:

/Clear 0 **def**

/Copy 1 **def**

/Sover 2 **def**

/Sin 3 **def**

/Sout 4 **def**

/Satop 5 **def**

/Dover 6 **def**

/Din 7 **def**

/Dout 8 **def**

/Datop 9 **def**

/Xor 10 **def**

/PlusD 11 **def**

/Highlight 12 **def**

/PlusL 13 **def**

Implementation Notes and Limitations

Partially Transparent Alpha

Alpha values that are not completely opaque (1) or completely transparent (0) should be used with caution. Compositing operations with partial transparency yield the highest image quality only when a large number of colors are available in the DPS color cube and gray ramp. That is, image

quality is best with a 24-bit TrueColor or 8-bit StaticGray visual, and image quality will be poor with an 8-bit PseudoColor visual. In addition, the performance of compositing operations is greatly reduced for partially transparent pixels due to the extra computation required in these cases.

Indexed Color Visuals

For best results with the **Highlight** *op*, the number of colors in the DPS context's gray ramp should be such that

```
fract(((float) numgrays - 1)* 2. / 3.) == 0
```

In other words, (numgrays = 4, 7, 6, 8, 16, ...). This ensures that the color 2/3 gray is not halftoned.

Given the limited number of colors usually available in the DPS color cube and gray ramp, images with alpha values that are not completely opaque (1) or completely transparent (0) should be avoided to obtain best image quality.

Compositing operations are only defined for pixels values that are in the gray ramp or color cube specified by the gstate. Compositing pixels with values outside the color cube and gray ramp may not yield expected results.

Monochrome Displays

The results of compositing operations for 1-bit drawables that have alpha values that are not equal to 0 or 1 is undefined.

The *op* **Highlight** inverts the color of the pixel on a 1-bit drawable.

Interaction with X Drawing Operations

Drawables that have been rendered to with non-opaque alpha have additional pixel storage associated with them, called the alpha channel. X Window system operations do not affect the alpha channel, with the following exceptions:

- When windows with alpha channel are exposed, if the window has an X background defined (background != None), when the background is painted, the alpha component of the exposed pixels is painted with alpha = 1.
- When a window is resized, the alpha channel storage is resized.

Destroying the Alpha Channel

The **erasepage** operator paints the current drawable of the graphics state with opaque white. Thus, the alpha values for all pixels in the drawable are equal to 1, and the alpha channel storage is destroyed.

Drawables with Unequal Depths

Compositing drawables with unequal depths is undefined.

2

Visuals on the Solaris X Server 3

This chapter discusses X window visuals on the Solaris X server. The chapter includes information on the following:

- Default visual
- Visuals on multi-depth devices
- Gamma-corrected visuals
- Hints on window programming with visuals

About Visuals

A display device can support one or more display formats. In the X window system, the display formats supported by the window server are communicated to client applications in the form of *visuals*. A visual is a data structure describing the display format a display device supports.

When an X11 client creates a window, it specifies the window's visual. The visual describes the display characteristics for each pixel in the window. In other words, a window's visual instructs the display device's video hardware how to interpret the value of the window's pixels.

For each display device configured into the system, there is an X11 screen. For each screen, a list of supported visuals is exported by the server to client applications. This list of visuals tells the client application which display formats are available for creating windows.

The visuals exported by the server for a display screen are not fixed; they depend on the screen's device handler. Since the exporting of visuals is under the control of the device handler, client applications must be prepared to deal with a wide variety of visuals, including visuals with depths other than those that have previously been common, such as 1, 8, and 24 bits. Visuals with depths of 4, 16, and odd depths may not potentially by exported, and clients must be prepared to handle them.

Client applications can query the list of supported visuals for a screen by calling the Xlib routines `XGetVisualInfo(3)` or `XMatchVisualInfo(3)`, and can query the list of supported visuals using the utility `xdpyinfo`(1). For general information on color and visuals in X11, see the X11 documentation listed in the preface to this manual.

Default Visual

For each X11 screen, one of the exported visuals for the screen is designated the *default visual*. The default visual is the visual assigned to the screen's root window, and this visual is the visual that most applications use to create their windows. When a client application starts, its windows are assigned the default visual unless the application specifies a different visual.

The *built-in default visual* is the visual hard-coded in the Solaris X server. For each screen, there is a default visual that depends on the characteristics of the display device for that screen. This is the default visual unless you specify a different default visual when you run `openwin(1)`.

Users can change the default visual that window server advertises in the connection block. One reason for this is to force client programs that cannot run in the default visual to run in a specific visual. For example, on a 24-bit device that has the TrueColor visual as its default visual, an application that cannot run with 24-bit color may run on a PseudoColor visual.

For developers on multi-depth devices, changing the default visual is a useful way to test that your application works in different configurations. For information on how to change the default visual, see the xsun(1) man page. The default visual and the list of supported visuals exported by the server can be examined from X11 using `XGetVisualInfo(3).`

Visuals on Multi-Depth Devices

The Solaris X server supports devices that can display windows of more than one pixel depth simultaneously. These devices are called *multi-depth* devices. Since most of these devices are implemented with separate groups of bit planes for each depth, the term *multiple plane group* (MPG) device is often used for these devices.

For each depth, there might be one or more visuals exported. For most MPG devices, windows can be created using any of the exported visuals. For applications that prefer a TrueColor visual, the developer should determine whether the TrueColor visual is available, since it may be available even if PseudoColor is the default visual.

Hints for Windows Programming With Visuals

This section discusses various issues that may arise when programming X11 applications for devices that support more than one visual.

Default Visual Assumptions

A common mistake in programming an X11 client is to assume that the default visual has an indexed class (for example, PseudoColor or StaticColor). It is possible for the default visual to be 24-bit TrueColor on some devices. Clients expecting to run on these devices must be prepared to handle this type of default visual.

Other common programming mistakes with visuals are:

- Assuming the default depth is 8
- Assuming the colormap is writable
- Using a default visual that is not appropriate rather than searching for an appropriate visual using `XGetVisualInfo`

If the device does not support a visual requested by a client, the following error message is returned. In this error message, # represents the depth number requested, and *n* represents the requested display device. If this message is returned for a supported visual / device combination as indicated in Table A-1 on page 102, then an installation problem exists.

```
Error: cannot provide a default depth # for device /dev/fbs/n
```

In general, client applications may need to be modified to make them more portable in the presence of different default visual types.

Setting the Border Pixel

When creating a window with a visual that is not the default visual, applications must set the `border_pixel` value in the window attribute structure, or a `BadMatch` error occurs. This is a common programming error that may be difficult to debug. For information on setting the border pixel, see the `XCreateWindow` man page.

Note – If you are experiencing improper graphics and double-buffering performance (such as lack of acceleration), OpenWindows might not have been installed as `root`.

Gamma-Corrected Visuals

The linearity attribute of a visual describes the intensity response of colors it displays. On a cathode ray tube (CRT) monitor, the colors displayed are actually darker than the colors requested. This darkening is caused by the physics of monitor construction. Some devices support visuals that compensate for this darkening effect. This is called *gamma correction*.

Gamma correction is done by altering colors coming out of the frame buffer with the inverse of the monitor's response. Because the overall intensity of a gamma-corrected visual is a straight line, a gamma corrected visual is called a *linear* visual; a visual that is not gamma corrected is called a *nonlinear* visual.

Linearity is not a standard X11 attribute for visuals. However, some applications require a linear visual to avoid visible artifacts. For example, a graphics application using antialiased lines may produce objectionable “roping” artifacts if it does not use a linear visual. This kind of application is called a *linear application*. An application requiring a nonlinear visual for best display of colors is called a *nonlinear application*. Most X11 applications are nonlinear applications.

On most devices, the linearity of default visuals is nonlinear. Therefore, linear applications should not depend on the default and should always explicitly search for a linear visual. Similarly, it is a good idea for nonlinear applications to explicitly search for a nonlinear visual. Since this is typically the default on most devices, it is not as critical, but it is still a good policy to do so.

To determine whether a visual is linear, applications can use the interface `XSolarisGetVisualGamma`(3). For more information on gamma correction, refer to *Fundamentals of Computer Graphics* by Foley and Van Dam.

Finding a Linear Visual

Linearity of a visual can be determined in Solaris by querying the visual’s gamma. This is done by calling `XSolarisGetVisualGamma`(3). If the gamma value is equal to (or close to) 1.0, the visual is linear. Otherwise, it is nonlinear. A good rule-of-thumb for the closeness tolerance is 10%. To use the `XSolarisGetVisualGamma` API, the application must be linked with the Solaris `libXmu`.

Code Example 3-1 on page 40 is an example of selecting the best visual for a typical XGL™ 3D linear application. In this example, the application uses a nonlinear visual if a linear one cannot be found. This is only one possible visual selection policy.

Note – If the gamma of any visual on the device is changed, either through reconfiguration or calibration, the window system should be restarted. Otherwise, applications using `XSolarisGetVisualGamma` that are already running will not detect the change and may use the wrong visual.

Code Example 3-1 3D Linear Visual Selection

```
/*
** Returns the visual of the given depth, class and linearity,
** or NULL if not found.
*/
Visual *
match_visual (Display *dpy, int screen, int depth, int class,
            Bool wantLinear)
{
 XVisualInfo template;
 XVisualInfo *vinfo, *vi;
 int nitems, isLinear, i;
 double gamma;

 template.screen = screen;
 template.depth = depth;
 template.class = class;
 if (!(vinfo = XGetVisualInfo(dpy, VisualScreenMask |
               VisualDepthMask | VisualClassMask,
               &template, &nitems)) || nitems <= 0) {
    return (NULL);
 }

 for (i = 0, vi = vinfo; i < nitems; i++, vi++) {
    if (XSolarisGetVisualGamma(dpy, screen, vi->visual, &gamma)
        == Success) {
     /*
     ** A good rule of thumb for linearity of a visual is
     ** whether the gamma is within 10% of 1.0.
     */
     isLinear = (gamma >= 0.9 && gamma <= 1.1);
     if ((wantLinear && isLinear) || (!wantLinear && !isLinear)) {
        Visual *visual = vi->visual;
        XFree(vinfo);
        return (visual);
     }
    }
 }

XFree(vinfo);
return (NULL);
}
```

Here is the main routine of the example:

```
main ()
{
    Visual vis;
 ...

 if ((vis = match_visual(display, screen, 24, TrueColor, True))) {
    fprintf(stderr, “Found a linear 24-bit TrueColor visual\n”);
    visualClass = TrueColor;
    depth = 24;
}
else if ((vis = match_visual(display, screen, 24, TrueColor,False))){
    fprintf(stderr, “Found a nonlinear 24-bit TrueColor visual\n”);
    visualClass = TrueColor;
    depth = 24;
}
else if ((vis = match_visual(display, screen, 8, PseudoColor,False))){
    fprintf(stderr, “Found a nonlinear 8-bit PseudoColor visual\n”);
    visualClass = PseudoColor;
    depth = 8;
}
else {
    fprintf(stderr, “Cannot match 24 or 8 bit visual\n”);
    exit(1);
}

 ...
}
```

Visual Selection Alternatives

The above code example illustrates only one possible visual selection policy. Other policies can be implemented. It is recommended that applications be written to handle a wide variety of visual configurations. Some devices, for example the GX, do not have any linear visuals. Other devices have only a single linear 24-bit TrueColor visual. Other types of devices may support both linear and nonlinear visuals at the same time. In general, the most prudent way to write a portable application is to deal gracefully with all these configurations. This may involve printing a warning message if the visual of the desired linearity is not found. Or, if a linear application cannot find a linear visual, a useful trick is to manually darken in the application the colors given

to X11. This is tantamount to performing your own gamma correction. The gamma value returned by `XSolarisGetVisualGamma` can be used to determine how much to darken the colors.

Note – `XSolarisGetVisualGamma` is a *Public* interface of Solaris and is fully supported. In the future, a color management system may also provide this functionality. When this occurs, this will become the preferred way of getting this information. But until then, `XSolarisGetVisualGamma` should be used. When this color management system is introduced, applications using `XSolarisGetVisualGamma` will continue to run with no modification and will actually benefit from the increased accuracy of the color management system.

Font Support 4

This chapter provides information on font support in the Solaris X server. The chapter includes information on the following topics:

- Available font formats
- Outline and bitmap fonts
- Location of fonts and changing the default font path

Font Support in the Solaris X Server

The Solaris X server provides font support in both the X11 server and the Display PostScript (DPS) extension. Font formats from numerous vendors can be used to display text in English or foreign languages, including Asian languages. Symbol fonts can be used to display mathematical equations. The Solaris environment provides 55 Latin fonts for European text and two symbol fonts. Other fonts can also be added to the system.

The Solaris X server can also be a client of the font server `fs`. The font server renders fonts for the X server. `fs` can be started manually or automatically. For more information on this command, see the `fs(1)` man page.

Available Font Formats

Fonts from different vendors come in different formats. Table 4-1 and Table 4-2 list the various font formats, their vendors, and the associated file types supported by the Solaris environment. Table 4-1 lists outline fonts; Table 4-2 lists bitmap fonts.

Table 4-1 Outline Font Formats

Font Format	Vendor	File Type
F3	SunSoft	`.f3b`
Type1 (ASCII)	Adobe and various foundries	`.pfa`
Type1 (binary)	Adobe and various foundries	`.pfb`
Type 3	Adobe and various foundries	`.ps`
Speedo	Bitstream	`.spd`

Table 4-2 Bitmap Font Formats

Font Format	Vendor	File Type
Portable compiled format	MIT	`.pcf`
Bitmap distribution format	Adobe	`.bdf`
Big Endian prebuilt format	Adobe	`.bepf`
Little Endian prebuilt format	Adobe (for x86 only)	`.lepf`

The fonts provided by the Solaris X server are located in the `/usr/openwin/lib/X11/fonts` directory. For more information on the directory structure, see “Locating Fonts” on page 48.

The Solaris environment is configured so that most X11 fonts are also available in DPS (see Table 4-3). DPS supports a slightly different set of fonts than those supported by X11.

Table 4-3 Font File Availability

Font Format	Available in X11	Available in DPS
F3	Yes	Yes
Type1 outline fonts-ASCII	Yes	Yes
Type1 outline fonts-binary	Yes	Yes
Type 3	Yes	Yes
Speedo	Yes	No
Portable compiled format	Yes	Yes
Bitmap distribution format	Yes	No
Big Endian prebuilt format	No	Yes
Little Endian prebuilt format	No	Yes

Associated Files

The Solaris environment provides files with these extensions. They are not intended to be edited.

- `.afm` Adobe Font Metrics files read by client for kerning information
- `.map` F3 files read by X11 and DPS for encoding purposes
- `.trans` F3 files read by DPS for composite font construction
- `.ps` PostScript Files for composite font and PostScript resource construction
- `.enc` Encoding files used by X11 and DPS
- `.upr` Display PostScript resource files

4

Outline and Bitmap Fonts

Solaris supports two types of font representation: *outline* fonts and *bitmap* fonts. In the X server, outline fonts can be scaled to any size; in Display PostScript they can also be rotated and skewed. To display a letter from an outline font, the server scales and rotates only the outline of the character. This repositioned outline is then *rendered* into pixel form (bitmap) for display on the screen. This rendered bitmap is also stored in the glyph cache for reuse.

Because certain font sizes occur frequently, they are also kept in separate files in prerendered bitmap form. This saves the server from having to scale and render them. However, the resulting bitmap fonts can be displayed in only one size and orientation. Some of these fonts have also been hand-tuned to look better and be more readable. As they are encountered, these bitmaps are also placed in the glyph cache. The recommended bitmap format is the portable compiled format (`.pcf`).

The `/usr/openwin/bin` directory contains the following tools to convert fonts between the outline and bitmap font representation, as well as between various bitmap formats. See the corresponding man pages for more detailed information.

- `makebdf` Creates bitmap distribution format files (`.bdf`) from outline font files (`.f3b`)
- `bdftopcf` Converts a font from `.bdf` format to portable compiled format (`.pcf`)

As illustrated in Table 4-4, many bitmap font file formats are architecture-dependent binary files. They cannot be shared between machines of different architectures (for example, between SPARC and x86).

Table 4-4 Bitmap Font Binaries

Font Format	Binary	Architecture-Specific
Bitmap distribution format	No	No
Portable compiled format	Yes	No
Little Endian prebuilt format	Yes	Yes (x86)
Big Endian prebuilt format	Yes	Yes (SPARC)

The Solaris environment contains compressed .pcf files (files with .pcf.Z extensions). You can uncompress these if you want. If you add fonts to your system, you can either compress the files or not. Use uncompressed files if you want the fonts to display somewhat faster. Leave the files compressed if you want to conserve disk space.

Replacing Outline Fonts With Bitmap Fonts

The Solaris environment automatically replaces some outline fonts with bitmap fonts when the size is appropriate. This improves performance, and in some cases improves the aesthetics and readability of the text. There may be several sizes at which replacement occurs for a given outline font.

Replacement Conditions

Currently in DPS, the .pcf bitmap format is substituted for F3 outline fonts and the .bepf (or .lepf) is substituted for Type1 fonts. Substitution occurs when there is no rotation, the requested pixel size is within one half of a pixel of the .pcf font size, and the .pcf font is an F3Bitmap resource in a .upr (PostScript resource) file.

Using F3 Fonts in DPS

F3 fonts behave exactly like Type1 fonts, except /FontType returns 7 instead of 1. For example, the following PostScript code works the same regardless of the kind of font.

```
/Helvetica findfont 50 scalefont setfont
10 10 moveto (ABC) show
```

But the following code yields 7 for an F3 font and 1 for a Type1 font.

```
currentfont /FontType get ==
```

The kind of font returned depends on the current DPS internal resource path. See “Changing the Resource Path in DPS” on page 50 for details.)

4

Locating Fonts

By default, the Solaris server looks for fonts in directories under the `/usr/openwin/lib/X11/fonts` directory. Table 4-5 shows the complete font directory structure. The directory names are preceded by `/usr/openwin/lib/X11/fonts`.

Table 4-5 Font Directory Structure

Directory	Subdirectory	File Suffixes	Contents
`/100dpi`		`.pcf`	Bitmap fonts
`/75dpi`		`.pcf`	Bitmap fonts
`/F3`	`/afm`	`.f3b`	F3 format outline fonts
	`/map`	`.map`	F3 character set specifications
`/F3bitmaps`		`.pcf`	Bitmap fonts
`/Speedo`		`.spd`	Bitstream Speedo format outline fonts
`/Type1`		`.pfa`, `.pfb`	Type1 outline fonts
	`/afm`	`.afm`	Adobe font metrics
	`/outline`	`.pfa`, `.pfb`	Type1 outline fonts
	`/prebuilt`	`.bepf`, `.lepf`	Bitmaps for SPARC Solaris and x86
`/Xt+`		`.pcf`	Bitmap fonts
`/Type3`		`.ps`	PostScript outline fonts
`/encodings`		`.enc`	Encodings
`/misc`		`.pcf`	Bitmap fonts

Changing the Default Font Path in X11

In X11, the default font path is:

```
/usr/openwin/lib/X11/fonts/F3,
/usr/openwin/lib/X11/fonts/F3bitmaps,
/usr/openwin/lib/X11/fonts/Type1,
/usr/openwin/lib/X11/fonts/Speedo,
/usr/openwin/lib/X11/fonts/misc,
```

```
/usr/openwin/lib/X11/fonts/75dpi,
/usr/openwin/lib/X11/fonts/100dpi,
/usr/openwin/lib/X11/fonts/Xt+
```

You can change the default font path either at startup or after the server has been started. At startup, use the following command to change the font path. In this command, the user-defined directory list is a comma separated list of directories for the server to search. Note that the directory paths *must* be absolute.

```
example% openwin -fp /<user-defined-directory-list>
```

After the server has started, you can change the font path using either the `xset` command or the Xlib function `XSetFontPath`. For `xset`, use only *one* of the following:

```
example% xset +fp /user-defined-directory-list
example% xset fp+ /user-defined-directory-list
example% xset fp- /user-defined-directory-list
```

The `xset +fp` command prepends */dir1/dir2/fonts* to the X11 font path.

```
example% xset +fp /dir1/dir2/fonts
```

The `xset fp+` command appends */dir1/dir2/fonts* to the X11 font path. The `xset fp-` command removes */dir1/dir2/fonts* from the X11 font path.

```
example% xset fp- /dir1/dir2/fonts
```

Note – Since `xset` dynamically changes the font path, you do not need to restart the server to change the default font path.

For more information on `xset`, see the `xset` man page; for information on `XSetFontPath`, see the *Xlib Reference Manual*.

Changing the Resource Path in DPS

In DPS, fonts are considered resources in the font category. Their associated files are specified by resource (`.upr`) files. DPS resource files reside in directories specified by the resource (font) path. This path is a list of directories maintained internally by DPS. DPS uses the default resource path specified by the `PSRESOURCEPATH` environment variable to initialize itself. If `PSRESOURCEPATH` is not defined, DPS uses `/usr/openwin/lib/X11`. (See *Programming the Display PostScript System with X* for further information on resource database files.)

Warning – Because DPS maintains so many internal font caches, you cannot remove a path from the DPS resource path. DPS appends all paths subsequent to the default path to the resource path, regardless of where they end up in the X11 font path. Thus fonts available in X windows might be different from those available in DPS. However, the DPS resource path is dynamic. Fonts should be accessible after the `xset` command completes. Any change to the X font path is passed to DPS. If there are `.upr` files present, DPS appends the font files to its internal resource path. The examples in the remainder of this section illustrate some of the DPS and X11 font path behavior.

As shown in "Changing the Default Font Path in X11" on page 48, use `xset +fp` to prepend */dir1/dir2/fonts* to the font path, and use `xset fp+` to append */dir1/dir2/fonts* to the font path. If there are any `.upr` files present, the `xset` command (with either `fp+` or `+fp`) also appends */dir1/dir2/fonts* to the DPS resource path.

The following command removes */dir1/dir2/fonts* from the X11 font path, but does not alter the DPS resource path.

```
example% xset fp- /dir1/dir2/fonts
```

The following `openwin` command appends */dir1/dir2/fonts* to the DPS resource path.

```
example% openwin -fp /dir1/dir2/fonts
```

Use the following `xset` command to set the X11 font path to */dir1/dir2/fonts* and to append */dir1/dir2/fonts* to the existing DPS resource path.

```
example% xset fp= /dir1/dir2/fonts
```

Note – To clear the X11 font path and the DPS resource path, exit OpenWindows and then restart.

Adding New Fonts

To add new bitmap and outline fonts to the OpenWindows Server, follow the steps outlined in the following sections. These instructions apply to one byte fonts, such as the ISO Latin-1 fonts for English and European character sets. Multibyte fonts, such as Kanji fonts, might require additional files from the font supplier.

1. **Create a directory for the new fonts.**
   Do not add fonts to existing font directories—you might corrupt files in those directories, and you also must be superuser.

   For this example, */newfonts* is the directory name.

```
example% mkdir /newfonts
```

2. **Copy or move all fonts to the */newfonts* directory.**

If you are installing bitmap fonts (`pcf`, `snf`, `bdf`, or `fb`), see additional steps in "Adding Bitmap Fonts." If you are installing outline fonts (`f3b`, `pfa`, `pfb`, or `spd` formats), see "Adding Outline Fonts" on page 53.

Adding Bitmap Fonts

Follow these steps if you are installing any of the bitmap font formats (`pcf`, `snf`, `bdf`, or `fb`).

1. **Use `mkfontdir` to create the `fonts.dir` file.**

```
example% cd /newfonts
example% /usr/openwin/bin/mkfontdir
```

   See the `mkfontdir(1)` man page for further details.

2. **If you want to define font "aliases," create a `fonts.alias` file.**
   Use this file to map the long internal XLFD font names to shorter names that are easier to enter on a command line. A sample `fonts.alias` file is shown below.

```
courier "-adobe-courier-medium-r-normal--0-0-0-0-m-0-iso8859-1"
courier-italic "-adobe-courier-medium-i-normal--0-0-0-0-m-0-iso8859-1"
courier-bold "-adobe-courier-bold-r-normal--0-0-0-0-m-0-iso8859-1"
courier-bolditalic "-adobe-courier-bold-i-normal--0-0-0-0-m-0-iso8859-
1"
```

   See the `mkfontdir(1)` man page for further details.

3. **Use `xset` to add the *`/newfonts`* directory to the server font path.**

```
example% xset fp+ /newfonts
```

   See the `xset(1)` man page for more information.

4. **Use `xlsfonts` to determine whether the server recognizes the new fonts.**
   `xlsfonts` lists all the names of all fonts that are accessible to the window server.

Adding Outline Fonts

Follow these steps to install outline fonts. The Solaris environment supports Type1 (`pfa`), Speedo (`spd`), and F3 (`f3b`) outline fonts.

Multibyte fonts might require additional files from the font supplier. For F3 format fonts, you need the `.map` and `.trans` files. The server also uses the `.map` file. It provides a mapping between the character name and its F3 code. The DPS extension uses the `.trans` file to support multiple byte encodings. It contains the definitions of these encodings.

1. **If you are installing Type1 (`pfa` or `pfb`) fonts or Type 3 (`ps`) fonts, run `makepsres` in the *`/newfonts`* directory.**
   This creates a `PSres.upr` file. The system requires this file if you want to use these fonts in X or DPS client applications.

2. **If you are installing F3 fonts, create a `.upr` file.**
   Use the template below for the `.upr` file. Replace the example values with values that reflect the fonts you want to add. Follow the syntax used in the example. Include the `//` before the directory name you want to install into. Use the `=` in the lines where you include the map file and font file names.

   If the `.map` file included with your font is not in `/usr/openwin/lib/X11/F3/map`, then include it in the `.upr` file.

```
PS-Resources-1.0          #mandatory
F3MapFile                 #put this in if you are adding map files
FontOutline               #put this in if you are installing F3 fonts
.                         #mandatory
//home/newfonts           #name of directory to install fonts in:
                          #// required (this is an example)
F3MapFile                 #put this in if you are adding map files
latin=map/latin.map       #name of the map file and where it is
                          located (this is an example)
.                         #put this in if you are adding map files
FontOutline               #put this in if you are installing F3 fonts
Helvetica=Helvetica.f3b   #put the font file name here (this is
                          an example)
Times-Roman=Times-Roman.f3b#put in as many font files as you want
.                         #mandatory
```

3. **If you are installing Type1 (`pfa` or `pfb`), Type 3 (`ps`), or Speedo (`spd`) fonts, create a `fonts.scale` file.**
   The `fonts.scale` file contains the mapping of an internal X11 font name to a known font name. The `fonts.scale` file is copied to the `fonts.dir` file automatically. Do not edit the `fonts.dir` file. Any changes you make are overwritten when you run `mkfontdir`.

   For example, here is a `fonts.scale` file for a directory containing four Type1 fonts.

```
cour.pfa -adobe-courier-medium-r-normal--0-0-0-0-m-0-iso8859-1
couri.pfa -adobe-courier-medium-i-normal--0-0-0-0-m-0-iso8859-1
courb.pfa -adobe-courier-bold-r-normal--0-0-0-0-m-0-iso8859-1
courbi.pfa -adobe-courier-bold-i-normal--0-0-0-0-m-0-iso8859-1
```

   See the `mkfontdir(1)` man page for more information on the `fonts.scale` file.

Note – X11 names must follow the standard XLFD font naming convention, using 0's in appropriate fields to indicate outline fonts. See the *X Protocol Reference Manual* for information on the XLFD font naming convention.

4. **Use `mkfontdir` to create the `fonts.dir` file.**
   If you are installing Type1 or Speedo fonts, your `fonts.scale` file is copied to `fonts.dir`.

```
example% cd /newfonts
example% /usr/openwin/bin/mkfontdir
```

   See the `mkfontdir(1)` man page for further details.

5. **If you want to define font "aliases," create a** `fonts.alias` **file.**
   Use this to map the long internal XLFD font names to shorter names that are easier to enter on the command line.

   Here is an example `fonts.alias` file.

```
courier "-adobe-courier-medium-r-normal--0-0-0-0-m-0-iso8859-1"
courier-italic "-adobe-courier-medium-i-normal--0-0-0-0-m-0-iso8859-1"
courier-bold "-adobe-courier-bold-r-normal--0-0-0-0-m-0-iso8859-1"
courier-bolditalic "-adobe-courier-bold-i-normal--0-0-0-0-m-0-iso8859-1"
```

   See the `mkfontdir(1)` man page for further details.

   If you are installing an F3 font and the character set supported by this font is *not* one of the following, the font supplier must provide an encoding file (`.enc` file).

   - iso8859-1
   - iso8859-2
   - symbol
   - jisx0201.1976-0
   - jisx0208.1983-0

6. **Copy the** `.enc` **file described in Step 5 (if you have one) to** `/usr/openwin/lib/X11/fonts/encodings`**, and add an entry for it in the** `encodings.dir` **file in the same directory.**

7. **Use** `xset` **to add the** */newfonts* **directory to your font path.**

```
example% xset fp+ /newfonts
```

   See the `xset(1)` man page for more information.

8. **Use** `xlsfonts` **to determine whether the server recognizes your new fonts.**
   `xlsfonts` lists the names of all fonts that are accessible to the window server.

For DPS, you can determine whether the server recognizes your fonts using the following commands:

```
example% dpsexec
PostScript(r) Version 2015.103
(c) Copyright 1984-1994 Adobe Systems Incorporated.
Typefaces (c) Copyright 1981 Linotype-Hell AG and/or its subsidiaries.
All Rights Reserved.
PS> /fontname findfont
```

You can now use the fonts in the */newfonts* directry in your applications. In X11, you do not need to restart the Solaris X server since `xset` dynamically changes the font path. See “Changing the Default Font Path in X11” on page 48 for more information.

Using OPEN LOOK Fonts on X Terminals

The `/usr/openwin/share/src/fonts` directory contains OPEN LOOK fonts in `bdf` format. Follow the instructions from your vendor on how to install the fonts.

Transparent Overlay Windows 5

This chapter presents information on the application programming interface (API) that provides transparent overlay window capabilities in the Solaris environment. The chapter includes information on the following topics:

- How overlay windows differ from standard X windows
- How to create and draw to overlay windows
- How to ensure that applications using the overlay window API are portable to a wide range of devices

What are Transparent Overlay Windows?

The transparent overlay extension allows the creation and manipulation of transparent overlay windows. These windows are X windows that allow the user to see through to the underlying window on a per-pixel basis. No special hardware is needed to create and use transparent overlay windows, as this functionality has been implemented in software. Complex transparent overlay manipulation on simple hardware may be time consuming; however, the X server can make use of special overlay hardware if available and the client chooses the correct visuals. Note that, depending on your hardware and needs, you may have to adapt the client color allocations for transparent overlay windows.

Overlay windows allow applications to display temporary imagery in a display window. Users of an application that provides transparent overlays can annotate an image with text or graphical figures, temporarily highlight certain

portions of the imagery, or animate figures that appear to move against the background of the imagery. When geometry in the overlay is cleared, any underlying graphics does not need to be regenerated.

The tranparent overlay extension allows the client to use standard X requests to draw primitives in *opaque paint*, which is a name for the standard way of drawing, or *transparent paint*, which makes affected pixels invisible. The paint type is associated with a standard X graphics context. Window backgrounds may also be set to transparent paint. Transparent overlay windows obey all regular window rules and operating proceedures. For example, a transparent overlay window can be positioned anywhere in the window stacking order, regardless of what hardware the windows are associated with. This is implemented in software with the Solaris X server multiple plane group (MPG) functionality.

The server's multiple plane group capability allows windows from different parts of the hardware to coexist. Each window is associated with a visual, which in turn is associated with hardware. Although some hardware is physically created such that there is a definite "layering" (for example, windows created in a hardware overlay plane might be expected to always be seen above the regular windows), MPG works around this limitation in software. MPG allows the stacking order of the windows to be unaffected by the physical imitations of the hardware. As a result, stacking is simply the same as in the standard server. If overlay hardware is available and requested, MPG takes care of minimizing the work and increasing preformance.

In general, an overlay is a pixel buffer (either physical or software simulated) into which graphics can be drawn. When the overlay is physical (that is, not simulated in software), erasing the overlay graphics does not damage the underlying graphics. This provides a performance advantage when the underlying graphics is complex and requires much time to repaint. When the overlay is in software, erasing the overlay graphics may generate an `Expose` event.

Basic Characteristics of Overlay Windows

An overlay window is a special class of an X `InputOutput` window into which pixels can be rendered transparently. Handles to overlay windows have the X window type `Window`. Just like standard X windows, overlay windows are drawables, and an overlay window handle can be passed to any Xlib drawing routine that takes a `Drawable`.

Overlay windows have extended the set of graphics context attributes to include an attribute for paint type. With the transparent overlay extension, transparent overlay windows can be rendered to with either opaque or transparent paint.

Overlay Window Paint Type

While standard X `InputOutput` windows and other drawables (such as pixmaps) accept only opaque paint, overlay windows permit pixels to be rendered with *transparent paint*. Valid pixel values painted opaquely obscure pixels in underlying windows. Such pixels have associated color values that are displayed. Pixels rendered transparently have no intrinsic color; they derive their displayed color from the pixels that lie beneath.

Valid pixel values for pixels painted opaquely are obtained via `XAllocColor()` or another standard pixel allocation mechanism. Painting opaquely with a non-valid pixel value, for example a value that falls outside the valid colormap entries for a visual, produces undefined results for both overlay windows and standard X `InputOutput` windows.

Paint type is defined with the data structure `XSolarisOvlPaintType`. By default, the paint type of a `GC` is opaque. The `XSolarisOvlPaintType` data structure is defined as:.

```
typedef enum {
    XSolarisOvlPaintTransparent,
    XSolarisOvlPaintOpaque,
} XSolarisOvlPaintType;
```

Overlay Window Viewability

An overlay window is considered viewable even if all its pixels are fully transparent. For viewable pixels in an overlay window that are fully transparent, the underlying pixels in the underlay will be displayed.

If an overlay window is unmapped or moved, the underlay beneath may receive exposure events. This, for example, is the case on devices that can not display the overlay window and underlay window in different plane groups.

Rendering Transparent Paint

Applications can render into overlay windows using Xlib primitives. In addition, applications can render transparent paint to overlay windows through a Solaris Visual graphics library, such as the XGL graphics library, by specifying in the `GC` for that library that the paint is to be transparent. Each Solaris Visual library has a defined way of rendering into a transparent overlay window. See the library's documentation for information.

More on Overlay Window Characteristics

In most respects, an overlay window is just like a standard X `InputOutput` window. Specifically, an overlay window has these characteristics:

- It can be mapped or unmapped. The routines `XMapWindow`, `XUnmapWindow`, `XMapSubwindows`, and `XUnmapSubwindows` apply.
- An overlay window can possess its own cursor or use its parent's cursor. In other words, `XDefineCursor` and `XUndefineCursor` apply to overlay windows.
- An overlay window appears in the output of `XQueryTree`.
- The `event_mask` and `do_not_propagate_mask` window attributes function normally. An overlay window can express interest in any type of event.
- `XTranslateCoordinates` and `XQueryPointer` apply to overlay windows.
- `save_under` applies as for standard X windows.
- `override_redirect` applies as for standard X windows.

An overlay window also has some characteristics that makes it unique as a window. The following sections describe these characteristics.

Overlay Window Background

As defined in the X specification, windows can have a *background*. The main purpose of window background is to display something in the exposed areas of a window in case the client is slow to repaint these areas. This background is rendered whenever the window receives an `Expose` event. The background is rendered before the `Expose` event is sent to the client. The background is also rendered when the client makes an `XClearArea` or `XClearWindow` request.

Like standard X `InputOutput` windows, overlay windows can also have a background. The background of an overlay window is rendered just like a non-overlay window in response to `Expose` events, `XClearArea` requests, or `XClearWindow` requests. In addition to the standard types of background (None, pixmap, pixel, or parent relative), overlay windows can also be assigned a new type of background: transparent. A new routine, `XSolarisOvlSetWindowTransparent`, is available to set the background type to transparent.

The background of an overlay window is transparent by default. However, the application can still specify one of the usual X types of background: `None`, a pixmap XID, a pixel value, or `ParentRelative`, as shown in Table 5-1.

Table 5-1 Background Values for an Overlay Window

Background	Description
transparent	Background of overlay window is transparent by default.
None	No rendering is performed when the overlay window encounters a condition that invokes background painting. Neither transparent nor opaque paint is rendered.
Pixmap ID	The background is rendered with opaque paint. The rendered pixel values are derived from the pixmap as defined in the X specification.

Table 5-1 Background Values for an Overlay Window

Background	Description
Single pixel value	The background is a solid color rendered with opaque paint.
ParentRelative	The behavior for a ParentRelative background depends on the parent window background and its type. If the parent window is an underlay, the background for the overlay window child will be rendered with opaque paint, and the rendered pixels will be as defined in the X specification. If the parent window is an overlay, the background of the overlay child will be the same as that of the parent, either transparent or opaque paint will be rendered.

Attempts to set the background of a non-overlay window with `XSolarisOvlSetTransparent` generates a `BadMatch` error. If an underlay window has a `ParentRelative` background and the parent window is an overlay with a transparent background, the underlay child is treated as if it has a background of `None`.

Overlay Window Border

The border of overlay windows is opaque. It is always drawn with opaque paint. Just like standard X `InputOutput` windows, the border width can be controlled with `XSetWindowBorderWidth`.

Overlay Window Backing Store

An overlay window can be granted backing store not only for the color information of its opaque pixels, but also for the paint type of its pixels. If the `backing_store` attribute of a window is set to `Always` or `WhenMapped`, the X11 server can grant backing store for an overlay window. When backing store is granted, both the color and paint information are retained.

The `backing_planes` and `backing_pixel` apply only to the color information of opaque pixels in the window.

Overlay Window Gravity

The bit and window gravity attributes (`bit_gravity` and `win_gravity`) apply to overlay windows. However, if the gravity calls for the movement of pixels, the transparency information is moved, along with the pixel color information.

Overlay Colormap

Overlay colormap installation follows the X rules. If your application uses pixel-sharing overlay/underlay pairs, create a single colormap for both windows. Refer to "Choosing Visuals for Overlay/Underlay Windows" on page 65 and "Designing an Application for Portability" on page 83 for more on the subject of pixel-sharing pairs.

If the pair is known never to share hardware color LUTs, different colormaps can be safely assigned to the overlay and underlay window without the occurrence of colormap flashing.

Note – To improve the portability of applications and to minimize color flashing, use colormaps with the same colors in both the overlay and underlay window colormaps. If this is not possible, use one of the visual inquiry routines to determine whether different colormaps can be assigned without producing flashing.

Input Distribution Model

Overlay windows can express interest in events just like a standard X window. An overlay window receives any event that occurs within its visible shape; the paint type of the pixel at which the event occurs doesn't matter. For example, if the window expresses interest in window enter events, when the pointer enters the window's visible shape, the window receives a window enter event, regardless of whether the pixel is opaque or transparent.

This has some implications for how applications should implement interactive *picking* (selection) of graphical objects. Applications that draw graphical figures into an overlay window above other graphical figures drawn into the underlay window should express interest in events in either the overlay or underlay

window, but not both. When the application receives an input event, it must use its knowledge of the overlay/underlay layering to determine which graphical figure has been picked.

For example, let's say the application expresses interest in events on the underlay window. When the application receives an event at coordinate (x, y), it should first determine if there is a graphical figure at that coordinate in the overlay. If so, the search is over. If not, the application should next see if there is a graphical figure at that coordinate in the underlay.

Print Capture

After graphical imagery has been rendered to an X window, the user may want the window contents to be captured and sent to a printer for hard copy output. The most widespread technique for doing this is to perform a *screen dump*, that is, to read back the window pixels with `XGetImage`, and to send the resulting image to the printer. To fit the image to the size of the printed page, some image resampling may be necessary. This can introduce *aliasing* artifacts into the image.

Another print capture technique that is growing in popularity in the X11 community is to re-render the graphics through a special printer graphics API. This API supports the standard Xlib graphics calls. It converts these calls into a page description language (PDL) format and sends it to the appropriate print spooler. The advantage of this technique is that the graphics can be scaled to fit the printed page by scaling the coordinates themselves rather than the pixels after scan conversion has been applied. As a result, aliasing artifacts are minimized.

The print API technique has a significant drawback when applied to an overlay/underlay window pair. Most PDLs only support the notion of opaque paint; they do not provide for the marking of transparent paint. In the PostScript PDL, for example, the marked pixels always supersede what was previously marked. Given such a limitation, it is not always possible to capture the imagery in an overlay/underlay window pair using this technique. Certainly, in applications where the background of the overlay is completely transparent and only opaque paint is drawn to it, the underlay could be marked first and the overlay marked second. But if transparent paint was drawn to the overlay, erasing other opaque paint in the overlay, this would not work.

Until this issue is resolved, capture overlay windows and send them to the printer using `XReadScreen` and resampling. Alternatively, do not use overlays to render information that is to be printed.

Choosing Visuals for Overlay/Underlay Windows

The Solaris transparent overlay API supports multiple plane group (MPG) and single plane group (SPG) devices. Display devices come in a wide variety of configurations. Some have multiple plane groups. Some have multiple hardware color lookup tables (LUTs). Some dedicate color LUTs to particular plane groups and some share color LUTs between plane groups. This wide variety makes it difficult for an application writer to construct portable overlay applications.

For a given type of underlay window, some devices can provide some types of overlay windows with high-performance rendering. Other devices provide the same type of overlay window but with slower rendering. Some devices can support overlays with many colors, and some devices cannot. Some devices can support simultaneous display of both overlay and underlay colors for all types of overlays and underlays. Others support simultaneous display of colors but not for all overlay/underlay combinations. Still others support a certain degree of simultaneous color display. These devices support more than one hardware color LUT. Hardware might not contain enough color LUTs to enable all applications to display their colors simultaneously.

The Solaris Visual Overlay Window API provides two utility routines to enable an application to negotiate with the system for a suitable overlay/underlay visual pair:

- `XSolarisOvlSelectPartner`
- `XSolarisOvlSelectPair`

These routines are described in the section “Designing an Application for Portability” on page 83.

The assumption is made that each application has an ideal configuration of windows and colors. An application should start out by asking for the “best” overlay/underlay pair. If this can be satisfied by the device, then the negotiation is complete, and the application proceeds to create windows on the selected underlay and overlay visuals. But if no visual pair satisfies the query, the application must relax its demands. To this end, it should specify the “next

best" pair. The application may choose to ask for less colorful visuals, or it may accept lower rendering performance on one of the visuals. The process continues until either a satisfactory visual is found, or the application decides it's not worth running in this environment without certain criteria being met.

The overlay API provides routines that enable the application to conduct such a negotiation in a single subroutine call. The application specifies criteria to be matched for either the overlay visual, the underlay visual, or both. Application programmers are encouraged to use these routines to ensure portability to the widest range of graphics devices.

Example Program

The program below demonstates a simple example of a transparent overlay. The program creates a transparent overlay window, draws the window border in white, displays a text string in white, and draws a white filled rectangle. The paint type is opaque by default, and the window background is transparent by default. Use the following `Makefile` to compile and link the program.

```
simple: simple.c
    cc -I../ -I/usr/openwin/include -o simple simple.c \
    -L/usr/openwin/lib -lX11 -lXext
```

Code Example 5-1 Transparent Overlay Example Program

```
#include <stdio.h>
#include <X11/Xlib.h>
#include "X11/Xmd.h"
#include <X11/extensions/transovl.h>
#include <X11/extensions/transovlstr.h>

Display               *display;
Window                window;
XSetWindowAttributes  attribs;

GC                    gc;
XGCValues             gcvalues;

main()
{
 display = XOpenDisplay("");
 attribs.override_redirect = True;
 attribs.border_pixel = WhitePixel(display, 0);
```

```
window = XSolarisOvlCreateWindow(display,
             DefaultRootWindow(display),
             100, 100, 500, 500, 10,
             CopyFromParent, InputOutput,CopyFromParent,
             CWBorderPixel | CWOverrideRedirect, &attribs);
gcvalues.font = XLoadFont(display, “fixed”);
gcvalues.foreground = WhitePixel(display, 0);
gc = XCreateGC(display, window, GCFont | GCForeground, &gcvalues);
XMapWindow(display, window);
XDrawString(display, window, gc, 50, 50, “This is a test”, 14);
XFillRectangle(display, window, gc, 70, 70, 100, 100);
XFlush(display);
while (1);
}
```

Overview of the Solaris Transparent Overlay Window API

The transparent overlay window API includes the routines listed in Table 5-2. These routines are provided by `libXext.so`. To use the Solaris overlay routines, do the following:

- Include the file `/usr/openwin/include/X11/extensions/transovl.h`
- Link the library device handler with the library `/usr/openwin/lib/libXext.so`

Table 5-2 List of Overlay Window Routines

Name	Description
`XSolarisOvlCreateWindow`	Creates an overlay window.
`XSolarisOvlIsOverlayWindow`	Indicates whether a window is an overlay window.
`XSolarisOvlSetPaintType`	Specifies the type of paint rendered by subsequent Xlib drawing.
`XSolarisOvlGetPaintType`	Gets the current paint type.
`XSolarisOvlSetWindowTransparent`	Sets the background state of an overlay window to be transparent.

Table 5-2 List of Overlay Window Routines

Name	Description
`XSolarisOvlCopyPaintType`	Renders opaque and transparent paint into the destination drawable based on the paint type attributes of the pixels in the source drawable.
`XSolarisOvlCopyAreaAndPaintType`	Copies the area and paint type from one pair of drawables to another.
`XReadScreen`	Returns the displayed colors in a rectangle of the screen.
`XSolarisOvlSelectPartner`	Returns the optimal overlay or underlay visual for an existing visual.
`XSolarisOvlSelectPair`	Selects an optimal overlay / underlay pair that best meets a set of defined criteria for the overlay and underlay visuals.

The remainder of this chapter discusses the transparent overlay API routines.

Creating Overlay Windows

You can create an overlay window using `XSolarisOvlCreateWindow`. This routine behaves exactly as `XCreateWindow` except that the resulting window is an overlay window. The newly created window can be rendered into with both opaque and transparent paint, and the background of the overlay window is transparent.

The `class` argument to `XSolarisOvlCreateWindow` should be `InputOutput`. An overlay window can be created as an `InputOnly` window but, in this case, it will behave like a standard `InputOnly` window. It is only for `InputOutput` windows that there is a difference between overlay and non-overlay.

The syntax and arguments for `XSolarisOvlCreateWindow` are shown below.

```
Window
XSolarisOvlCreateWindow(Display *display, Window parent, int x, int y,
       unsigned int width, unsigned int height,
       unsigned int border_width, int depth, unsigned int class,
       Visual * visual, unsigned long valuemask,
       XSetWindowAttributes * attr)
```

The arguments for this routine are the same as those for `XCreateWindow`.

`display`	Specifies the connection to the X server.
`parent`	Specifies the parent window.
`x, y`	Specifies the coordinates of the upper-left pixel of this window, relative to the parent window.
`width, height`	Specifies the width and height, in pixels, of the window.
`border_width`	Specifies the width, in pixels, of the window's borders.
`depth`	Specifies the depth of the window.
`class`	Specifies the class of the window. If the class is not `InputOutput`, the window will not be an overlay window.
`visual`	Specifies a pointer to the visual structure for this window.
`valuemask`	Specifies which window attributes are defined in the `attr` argument.
`attr`	Specifies the attributes of the window.

You can use any visual to create the overlay. However, not all overlay/underlay visual pairs may be optimal. Each screen defines a set of optimal overlay/underlay visual pairs. These define the optimal visuals of the overlay windows that can be created with a particular underlay visual.

Likewise, they define the optimal visuals of underlay windows that can be created with a particular overlay visual. You can determine the optimal pairs using `XSolarisOvlSelectPair` and `XSolarisOvlSelectPartner`.

The definition of *optimal* varies from device to device, but it will usually refer to the ability of a device to create an overlay window in a different plane group than that of an underlay window. See page 89 for more information on overlay/underlay visual pairs.

Overlay windows are destroyed with the Xlib routines `XDestroyWindow` or `XDestroySubwindows`.

Setting the Paint Type of a Graphics Context

You can set a `GC`'s paint type with the `XSolarisOvlSetPaintType` routine. `XSolarisOvlSetPaintType` specifies the type of paint rendered by subsequent Xlib drawing with the given `GC`. It controls whether Xlib drawing routines using this `GC` produce opaque or transparent pixels on overlay windows. The paint type specified applies to the `GC` until it is changed by another call to this routine. The paint type attribute applies to both the foreground and background `GC` attributes. The syntax and arguments are shown below.

```
void
XSolarisOvlSetPaintType (Display *display, GC gc,
       XSolarisOvlPaintType paintType)
```

`display`	Specifies the connection to the X server.
`gc`	Specifies the affected `GC`.
`paintType`	Specifies the type of paint rendered by subsequent Xlib drawing routines using the specified `GC`.

The value of `paintType` can be `XSolarisOvlPaintOpaque` or `XSolarisOvlPaintTransparent`.

- If the value of `paintType` is `XSolarisOvlPaintOpaque`, the pixels generated by subsequent Xlib drawing routines with this `GC` will be opaque. This means the pixels will obscure underlying pixels. This is the default.

- If the value of `paintType` is `XSolarisOvlPaintTransparent`, the pixels generated by subsequent Xlib drawing routines with this `GC` will be transparent. This means that, for these pixels, the color of the underlying pixels is displayed.

Setting the Background State of an Overlay Window

You can set the background state of an overlay window to be transparent with the `XSolarisOvlSetWindowTransparent` routine. Any background rendering that occurs after this request causes the background to be transparent. To change background state to any other value, use `XChangeWindowAttributes()`, `XSetWindowBackground()`, or `XSetWindowBackgroundPixmap()`.

The syntax and arguments of `XSolarisOvlSetWindowTransparent` are shown below.

```
void
XSolarisOvlSetWindowTransparent (Display *display, Window w)
```

`display`	Specifies the connection to the X server.
`w`	The overlay window.

Note – If `w` is not an overlay window, a `BadMatch` error results.

Rendering to an Overlay Window

Once an overlay window is created, you can use all the standard Xlib primitive rendering routines, such as `XDrawLines` and `XFillRectangles`, to draw into the window. When drawing to overlay windows, the paint type attribute of the `GC` is used to control the quality of the pixels rendered. The paint type attribute applies to both the foreground and background `GC` attributes. To set the paint type, use the `XSolarisOvlSetPaintType` routine; for information on this routine, see page 70.

The paint type of the GC also controls the type of pixels rendered with XPutImage. If the paint type of the argument GC is XSolarisOvlPaintOpaque, the color information from the source image is used and the pixels are rendered with opaque paint. However, if the paint type is XSolarisOvlPaintTransparent, the source color information is ignored, and the pixels are rendered with transparent paint.

If a GC with a paint type of XSolarisOvlPaintTransparent is used to render to a drawable other than an overlay window, such as an underlay window or pixmap, the GC paint type is ignored, and the pixels are rendered with opaque paint.

Querying the Characteristics of an Overlay Window

You can determine whether a window is an overlay window using the routine XSolarisOvlIsOverlayWindow. You can also determine a GC's current paint type using the routine XSolarisOvlGetPaintType.

Determining Whether a Window is an Overlay Window

You can use the routine XSolarisOvlIsOverlayWindow to determine whether a window is an overlay window. The routine returns True if the given window w is an overlay window and returns False otherwise.

```
Bool XSolarisOvlIsOverlayWindow (Display *display, Window w)
```

display	Specifies the connection to the X server.
w	Specifies the window.

Determining the Paint Type of a Graphics Context

The routine XSolarisOvlGetPaintType returns the GC's current paint type.

```
XSolarisOvlPaintType
XSolarisOvlGetPaintType (Display *display, GC gc)
```

`display`	Specifies the connection to the X server.
`gc`	The `GC` to be inquired about.

Pixel Transfer Routines

The Solaris overlay API provides three pixel transfer routines:

- `XSolarisOvlCopyPaintType` – Renders opaque and transparent point into a destination drawable based on the paint type attributes of the source drawable.
- `XSolarisCopyAreaAndPaintType` – Copies an area and its paint type from one pair of drawables to another.
- `XReadScreen` – Returns the colors displayed in a given area of the screen.

The existing Xlib pixel transfer routines `XGetImage`, `XCopyArea`, and `XCopyPlane` can also be used with overlay windows. The use of these routines is described below.

Filling an Area Using the Source Area Paint Type

The `XSolarisOvlCopyPaintType` routine uses the paint type information of a specified rectangle in a source rectangle to control a fill operation in a specified rectangle in a destination rectangle. The source rectangle and destination rectangle can be any type of drawable. If the source rectangle is an overlay, the paint type attribute of its pixels is used as the source of the copy, and the color information is ignored. If the source rectangle is any other type of drawable, the bit plane specified in the routine is treated as if it were paint type data and it is used for the copy. In this case, the bit plane must have only one bit set.

The syntax and arguments are shown below.

```
void
XSolarisOvlCopyPaintType(Display *display, Drawable src,
       Drawable dst, GC gc, int src_x, int src_y,
       unsigned int width, unsigned int height, int dest_x,
       int dest_y, unsigned long action, unsigned long plane)
```

`display`	Specifies the connection to the X server.
`src`	Specifies the source drawable from which to obtain the paint type information.
`dst`	Specifies the destination drawable.
`gc`	Specifies the `GC`.
`src_x`, `src_y`	Specify the x and y coordinates of the upper-left corner of the source rectangle relative to the origin of the source drawable.
`width`, `height`	Specify the width and height of both the source and destination rectangles.
`dest_x`, `dest_y`	Specify the x and y coordinates of the upper-left corner of the destination rectangle relative to the origin of the destination drawable.
`action`	Specifies which paint type data is to be copied. This can be one of `XSolarisOvlCopyOpaque`, `XSolarisOvlCopyTransparent`, or `XSolarisOvlCopyAll`.
`plane`	Specifies the bit-plane of the `src` drawable to be used as paint type information when the source is not an overlay.

`src` and `dst` must have the same screen, or a `BadMatch` error results.

Table 5-3 summarizes the possible combinations of `src` and `dst` and their actions. The left side of the table shows the possible `src` combinations. The top of the table shows the possible `dst` combinations. The actions A1-A4 are explained following the table.

Table 5-3 XSolarisOvlCopyPaintType Source/Destination Combinations and Actions

Source/Destination	Overlay	Drawable
overlay	A1	A2
drawable	A3	A4

- A1—Opaque pixels in the source overlay cause the corresponding pixels in the destination to be filled with opaque color as specified by the fill attributes of the `GC`. Transparent pixels in the source cause the corresponding pixels in the destination to be filled with transparent paint.
- A2—Opaque pixels in the source overlay cause the corresponding pixels in the destination to be filled according to the fill attributes of the `GC`. Transparent pixels in the source overlay cause the corresponding pixels in the destination to be filled according to the same fill attributes of the `GC`, but with the foreground and background pixels swapped.
- A3—The pixels in the destination overlay are filled with opaque paint or made transparent as in A1 above depending on the bit values of the source drawable's `plane`. Bit values of 1 in the source are treated as if they were opaque pixels and bit values of 0 are treated as if they were transparent.
- A4—The pixels in the destination drawable are filled with paint as in A2 above depending on the bit values of the source drawable's plane. Bit values of 1 in the source bit plane are treated as if they were opaque pixels and bit values of 0 are treated as if they were transparent.

The `action` argument specifies whether opaque paint (`XSolarisOvlCopyOpaque`), transparent paint (`XSolarisOvlCopyTransparent`), or both (`XSolarisOvlCopyAll`) should be operated upon. This allows a client to *accumulate* opaque or transparent paint.

If portions of the source rectangle are obscured or are outside the boundaries of the source drawable, the server generates `Expose` events, using the same semantics as `XCopyArea`.

This routine uses these GC components: function, plane-mask, fill-style, subwindow-mode, graphics-exposures, clip-x-origin, clip-y-origin, and clip-mask. It might use these GC mode-dependent components: foreground, background, tile, stipple, tile-stipple-x-origin, tile-stipple-y-origin.

XSolarisOvlCopyPaintType can generate BadDrawable, BadGC, BadMatch, and BadValue errors.

Copying an Area and Its Paint Type

The XSolarisCopyAreaAndPaintType routine copies the specified area of source drawable for the color information to the specified area of destination drawable for color information. If the destination drawable is not an overlay, it also fills the specified areas of paint type information destination drawable according to the paint type information specified in the paint type information source drawable.

You can use XSolarisOvlCopyAreaAndPaintType to combine an image in the client's memory space (consisting of color and/or paint type information) with a rectangle of the specified overlay window. To do this, first move the image and paint type data into the server: use XPutImage to copy the data into two pixmaps of the appropriate depths. Then call XSolarisOvlCopyAreaAndPaintType with the color and paint type drawables to copy information to the overlay.

You can also use XSolarisOvlCopyAreaAndPaintType to retrieve pixel information (color and/or paint type information) from a specified drawable. To do this, call XSolarisOvlCopyAreaAndPaintType with two separable destination drawables. To get the data from the server into the client's memory space, call XGetImage on each of the drawables.

The syntax and arguments for `XSolarisCopyAreaAndPaintType` are shown below.

```
void
XSolarisOvlCopyAreaAndPaintType(Display * display, Drawable
colorsrc,
       Drawable painttypesrc, Drawable colordst,
       Drawable painttypedst, GC colorgc, GC painttypegc,
       int colorsrc_x, int colorsrc_y, int painttypesrc_x,
       int painttypesrc_y, unsigned int width, unsigned int height,
       int colordst_x, int colordst_y, int painttypedst_x,
       int painttypedst_y, unsigned long action, unsigned long plane)
```

`display`	Specifies the connection to the X server.
`colorsrc`	The color information source drawable. `colorsrc` can be any depth drawable or an overlay window.
`painttypesrc`	The paint type information source drawable. `painttypesrc` can be any drawable or an overlay window. If `painttypesrc` is not an overlay window, the bit plane of `painttypesrc` specified in `plane` is treated as if it were paint type data and it is used for the copy. `plane` must have only one bit set in this case.
`colordst`	The color information destination drawable.
`painttypedst`	The paint type information destination drawable. If `colordst` is an overlay, this drawable will be ignored.
`colorgc`	The `GC` to use for the color information copy.
`painttypegc`	The `GC` to use to fill areas in `painttypedst`. If `colordst`/`painttypedst` is an overlay, this `GC` will be ignored.
`colorsrc_x` `colorsrc_y`	The X and Y coordinates of the upper-left corner of the source rectangle for color information relative to the origin of the color source drawable.

`painttypesrc_x` `painttypesrc_y`	The X and Y coordinates of the upper-left corner of the source rectangle for paint type information relative to the origin of the paint type source drawable.
`width, height`	The dimensions in pixels of all the source and destination rectangles.
`colordst_x` `colordst_y`	The X and Y coordinates of the upper-left corner of the destination rectangle for color information relative to the origin of the color destination drawable.
`painttypedst_x` `painttypedst_y`	The X and Y coordinates of the upper-left corner of the destination rectangle for paint type information relative to the origin of the paint type destination drawable. If `colordst`/`painttypedst` is an overlay, colordst_x and colordst_y will be used.
`action`	Specifies which paint type data is to be copied. This can be one of XSolarisOvlCopyOpaque, XSolarisOvlCopyTransparent, or XSolarisOvlCopyAll.
`plane`	Specifies the source bit-plane in `painttypesrc` to be used as paint type information when `painttypesrc` is not an overlay.

`colordst` can be any drawable, but must be of the same depth and have the same root as `colorsrc`, otherwise, a `BadMatch` error results. If `colordst` is an overlay, then `painttypedst` is ignored, otherwise `painttypedst` can be any type of drawable.

Table 5-4 summarizes the possible combinations of sources and destinations and their respective actions. The left side of the table shows the possible `colorsrc/painttypesrc` combinations and the top of the table shows the possible `colordst/painttypedst` combinations. The actions A1-A8 are

explained below the table. An Impossible entry in the table indicates that the given combination is impossible, since the `painttypedst` is ignored when the `colordst` is an overlay.

Table 5-4 XSolarisOvlCopyAreaAndPaintType Source / Destination Combinations and Actions

	Overlay/Overlay	**Overlay/Drawable**	**Drawable/Overlay**	**Drawable/Drawable**
overlay/overlay	A1	Impossible	A5	A5
overlay/drawable	A2	Impossible	A6	A6
drawable/overlay	A3	Impossible	A7	A7
drawable/drawable	A4	Impossible	A8	A8

- A1—The paint type information from `painttypesrc` is used as a mask to copy the color information from `colorsrc` to `colordst`. Opaque pixels in `painttypesrc` cause the corresponding pixel in `colorsrc` to be copied to `colordst`, transparent pixels cause the corresponding pixel in `colordst` to be made transparent. If a transparent pixel from `colorsrc` is copied to `colordst`, the actual color transferred will be undefined.
- A2—Same as A1 except that the paint type information is extracted from the bit-`plane` of `painttypesrc` specified by `plane`. A bit value of 1 indicates an opaque pixel whereas a bit value of 0 indicates transparent.
- A3—Same as A1 except that a non-overlay drawable is used to obtain the color information so there will be no undefined colors due to transparent pixels.
- A4—Same as A3 except that the paint type information is taken from the specified bit-plane of `painttypesrc` as in A2.
- A5—The paint type information from `painttypesrc` is used as a mask to copy the color information from `colorsrc` to `colordst` as in A1. In addition, the paint type information controls rendering to the `painttypedst` drawable as in `XSolarisOvlCopyPaintType`.
- A6—Same as A5 except that the paint type information is taken from the specified bit-plane of `painttypesrc` as in A2.
- A7—Same as A5 except that there will be no undefined colors due to transparent color source pixels.
- A8—Same as A7 except that the paint type information is taken from the specified bit-plane of `painttypesrc` as in A2.

The action argument specifies whether opaque paint (XSolarisOvlCopyOpaque), transparent paint (XSolarisOvlCopyTransparent), or both (XSolarisOvlCopyAll) should be copied. This allows a client to accumulate opaque or transparent paint.

NoExpose and GraphicsExpose events are generated in the same manner as XSolarisOvlCopyPaintType.

If an overlay is used for the colordst argument, the painttypedst, painttypegc, painttypedst_x and painttypedst_y arguments will all be ignored. A NULL pointer can be used for painttypegc and a value of None can be used for painttypedst. The overlay will have the exact paint type defined by the pixels in the area specified in painttypesrc. The color information copy will not affect the destination paint type.

This function uses these GC components from colorgc: function, plane-mask, subwindow-mode, graphics-exposures, clip-x-origin, clip-y-origin, and clip-mask.

If colordst is not an overlay then this function will use these GC components from painttypegc: function, plane-mask, fill-style, subwindow-mode, clip-x-origin, clip-y-origin, and clip-mask. In addition, it may also use these GC mode-dependent components: foreground, background, tile, stipple, tile-stipple-x-origin, and tile-stipple-y-origin.

XSolarisOvlCopyAreaAndPaintType can generate BadDrawable, BadGC, BadMatch, and BadValue errors.

Retrieving Overlay Color Information

The routine XReadScreen returns the displayed colors in a rectangle of the screen. It thus provides access to the colors displayed on the screen of the given window.

On some types of advanced display devices, the displayed colors can be a composite of the data contained in several different frame stores, and these frame stores can be of different depth and visual types. In addition, there can be overlay/underlay window pairs in which part of the underlay is visible beneath the overlay. Because the data returned by XGetImage is undefined for portions of the rectangle that have different depths, XGetImage is inadequate to return the picture the user is actually seeing on the screen. In addition,

`XGetImage` cannot composite pixel information for an overlay/underlay window pair because the pixel information lies in different drawables. `XReadScreen` addresses these problems.

Rather than returning pixel information, `XReadScreen` returns color information—the actual displayed colors visible on the screen. The routine returns the color information from any window within the boundaries of the specified rectangle. Unlike `XGetImage`, the returned contents of visible regions of inferior or overlapping windows of a different depth than the specified window's depth are not undefined. Instead, the actual displayed colors for these windows is returned.

Note – The colors returned are the ones that would be displayed if an unlimited number of hardware color LUTs were available on the screen. Thus, the colors returned are the `theoretical` display colors. If colormap flashing is present on the screen because there aren't enough hardware color LUTs to display all of the software colormaps simultaneously, the returned colors may be different from the colors that are actually displayed.

The syntax and arguments for this routine are shown below.

```
XImage *
XReadScreen (Display *display, Window w, int x, int y,
       unsigned int width, unsigned int height,
       Bool includeCursor)
```

`display`	Specifies the connection to the X server.
`w`	Specifies the window from whose screen the data is read.
`x`, `y`	Specify the X and Y coordinates of the upper-left corner of the rectangle relative to the origin of the window `w`.
`width`, `height`	Specify the width and height of the rectangle.
`includeCursor`	Specifies whether the cursor image is to be included in the colors returned.

If `w` is an overlay window, the overlay color information is returned wherever there is opaque paint in the specified rectangle. The color information of the underlay is returned wherever there is transparent paint in the overlay. In general, since this underlay can be an overlay window containing transparent paint, the color information for a coordinate (x, y) that contains transparent paint is the youngest non-inferior that has opaque paint at (x, y).

The color data is returned as an `XImage` structure. The returned image has the same width and height as the arguments specified. The format of the image is `ZPixmap`. The depth of the image is 24 and the bits_per_pixel is 32. The most significant 8 bits of color information for each color channel (red, green, blue) are returned in the bit positions defined by `red_mask`, `green_mask`, and `blue_mask` in the XImage. The values of the following attributes of the `XImage` are server dependent: `byte_order`, `bitmap_unit`, `bitmap_bit_order`, `bitmap_pad`, `bytes_per_line`, `red_mask`, `green_mask`, `blue_mask`.

If `includeCursor` is True, the cursor image is included in the returned colors. Otherwise, it is excluded.

Note that the borders of the argument window (and other windows) can be included and read with this request.

If a problem occurs, `XReadScreen` returns `NULL`.

Using Existing Xlib Pixel Transfer Routines With Overlay Windows

The Xlib pixel transfer routines `XGetImage`, `XCopyArea`, and `XCopyPlane` can also be used with overlay windows.

XGetImage

On non-overlay drawables, the `XGetImage` routine works as defined in the X11 specification. The same is true for overlay windows, with the exception that, on these windows, the color information returned for transparent pixels is undefined. Clients who simply want to retrieve the display colors for a region on the screen should use `XReadScreen`.

XCopyArea and XCopyPlane

When both the source and destination drawables are non-overlay, the `XCopyArea` and `XCopyPlane` routines work as defined in the X11 specification. However, note the following for the cases in which either the source or the destination drawable is an overlay window.

- When the source drawable is overlay and the destination drawable is non-overlay, only the color information is copied; the paint type information in the source is ignored. Color information for transparent pixels is undefined.
- When the source drawable is non-overlay and the destination drawable is overlay, the copy is performed as the paint type in the `GC` indicates. If the paint type is `XSolarisOvlPaintOpaque`, the color information is copied into the destination with opaque paint. If the paint type is `XSolarisOvlPaintTransparent`, the color information is ignored, and the destination pixels are transparent.
- When both the source drawable and destination drawable are overlay, the paint type of the source is ignored, and this behaves as if the source were not an overlay. If copying both color and paint type information is the desired result, use `XSolarisOvlCopyAreaAndPaintType`.

Designing an Application for Portability

The Solaris overlay API provides two routines that help ensure application portability across devices. These routines are:

- `XSolarisOvlSelectPartner` – Enables the application to select the visual that is the best partner for an existing overlay or underlay visual.
- `XSolarisOvlSelectPair` – Enables the application to select the optimal overlay and underlay visual pair from the set of all visual pairs for the screen.

These routines are described below.

Selecting a Visual for an Overlay/Underlay Window

Portable applications using overlays can search for an appropriate overlay visual to use for a given underlay visual, or vice versa. Each X screen supporting the overlay extension defines a set of overlay visuals whose windows are best for use as children of underlay windows. For each underlay visual, there is a set of optimal overlay visuals. Together, all combinations of underlay visuals and their optimal overlay visuals form the set of optimal overlay / underlay pairs for that screen. The overlay and underlay visuals of an optimal pair are partners of each other.

The routine `XSolarisOvlSelectPartner` allows the client to select, given an underlay visual, an optimal overlay that meets certain criteria. Inversely, it also allows the client to select an optimal underlay visual given an overlay visual. The client is assured that, short of X errors not related to overlays, it can successfully create a window with the returned visual.

This routine searches through the optimal partners of the given visual, applying the criteria specified. It returns a success or failure status depending on whether it finds a visual that meets the criteria. A criterion can be one of two types:

1. Hard criterion – A criterion that must be satisfied. Only visuals that meet hard criteria are candidates for successful matches.

2. Soft criterion – A desirable criterion, but one that is not required.

The visual that matches all hard criteria and the most soft criteria is chosen, and its attributes are returned. If two or more visuals are found that meet all of the hard criteria and the same number of soft criteria, one of them will be chosen and returned. It is implementation dependent which one is chosen.

The syntax and arguments for `XSolarisOvlSelectPartner` are shown below.

```
XSolarisOvlSelectStatus
XSolarisOvlSelectPartner (Display *display, int screen,
        VisualID vid, XSolarisOvlSelectType seltype, int numCriteria,
        XSolarisOvlVisualCriteria *pCriteria,
        XVisualInfo *visinfoReturn,
        unsigned long *unmetCriteriaReturn)
```

`display`	Specifies the connection to the X server.
`screen`	An integer specifying the screen for the visual `vid`.
`vid`	The XID of the visual to find a partner for.
`seltype`	The type of selection that is to be done.
`numCriteria`	The number of `XSolarisOvlVisualCriteria` structures in the `pCriteria` array.
`pCriteria`	An array of criteria structures in priority order from high to low specifying the criteria to be used in selecting the visual.
`visinfoReturn`	A pointer to a caller provided `XVisualInfo` structure. On successful return, this structure contains a description of the chosen visual.
`unmetCriteriaReturn`	A pointer to a bitmask that describes the criteria that were not satisfied. This return argument is meaningful only when the routine returns a value of `XSolarisOvlQualifiedSuccess`, or `XSolarisOvlCriteriaFailure`.

Argument Types

`XSolarisOvlSelectType` is an enumeration defining two types of selections that can be done in `XSolarisOvlSelectPartner`. It is defined as:

```
typedef enum {
    XSolarisOvlSelectBestOverlay,
    XSolarisOvlSelectBestUnderlay,
} XSolarisOvlSelectType;
```

`XSolarisOvlVisualCriteria` is a structure defining various criteria to be used during visual selection, along with indications of the stringency of the criteria. This structure is defined as:

```
typedef struct {
    unsigned long      hardCriteriaMask;
    unsigned long      softCriteriaMask
    int                c_class;
    unsigned int       depth;
    unsigned int       minColors;
    unsigned int       minRed;
    unsigned int       minGreen;
    unsigned int       minBlue;
    unsigned int       minBitsPerRGB;
    unsigned int       minBuffers;
} XSolarisOvlVisualCriteria;
```

`hardCriteriaMask` and `softCriteriaMask` are bitmasks whose values can be the logical OR of any of the following bitmasks:

```
#define XSolarisOvlVisualClass       (1L<<0)
#define XSolarisOvlDepth             (1L<<1)
#define XSolarisOvlMinColors         (1L<<2)
#define XSolarisOvlMinRed            (1L<<3)
#define XSolarisOvlMinGreen          (1L<<4)
#define XSolarisOvlMinBlue           (1L<<5)
#define XSolarisOvlMinBitsPerRGB     (1L<<6)
#define XSolarisOvlMinBuffers        (1L<<7)
#define XSolarisOvlUnsharedPixels    (1L<<8)
#define XSolarisOvlUnsharedColors    (1L<<9)
#define XSolarisOvlPreferredPartner  (1L<<10)
```

Return Types

`XSolarisOvlSelectStatus` is a value that indicates whether the routine succeeded in finding a visual and, if it failed, the reason for the failure. The return value can be one of:

```
typedef enum {
    XSolarisOvlSuccess,
    XSolarisOvlQualifiedSuccess,
    XSolarisOvlCriteriaFailure,
    XSolarisOvlFailure,
} XSolarisOvlSelectStatus;
```

- `XSolarisOvlSuccess` is returned if the search is completely successful in finding a visual that meets all hard and soft criteria of one of the `XSolarisOvlVisualCriteria` structure.
- `XSolarisOvlQualifiedSuccess` is returned if the chosen visual satisfies all hard criteria of one of the `XSolarisOvlVisualCriteria` structure, but doesn't meet all soft criteria. In this case, `unmetCriteriaReturn` contains the logical OR of the soft criteria that were not met.
- `XSolarisOvlCriteriaFailure` indicates that no visual could be found that meets all the hard criteria of any of the `XSolarisOvlVisualCriteria` structures. In this case, `unmetCriteriaReturn` contains the logical OR of the hard criteria that were not met for the XSolarisOvlVisualCriteria structure with the fewest hard criteria not met.
- `XSolarisOvlFailure` is returned if some other error is encountered besides criteria match failure.

Multiple Criteria Sets

`XSolarisOvlSelectPartner` supports a *degradation sequence* of criteria sets. This means that multiple criteria sets can be specified in a single call. First, the routine attempts to find a visual matching the first criteria set. If a visual is found that meets all of the hard criteria of the first set, this visual is chosen. If no visual meets all hard criteria of the first set, the routine performs a search using the second criteria set. This process continues until either a visual is found that meets the hard criteria of some criteria set, or all sets have been

used to search. This degradation sequence allows clients to specify the criteria for the most preferred visual as the first criteria set. Visuals that are acceptable but are less desirable can be specified in criteria sets following the first criteria set. This allows the search to proceed through a progressive relaxation in the client's requirements for the visual with a single subroutine call.

Any of the possible criteria can be specified either as a hard or soft criteria for a particular criteria set. For a given set, `hardCriteriaMask` is the logical OR of the criteria bitmasks that are to be applied as hard criteria during the search. Likewise, `softCriteriaMask` is the logical OR of the soft criteria bitmasks.

Some criteria have values associated with them. These values are provided by other data members in the `XSolarisOvlVisualCriteria` structure. In the criteria descriptions that follow, these data members are mentioned where applicable.

- `XSolarisOvlVisualClass` specifies that the client wants the selected visual to have a specific visual class. The required class is specified in `c_class`.
- The following criteria interact within one another: `XSolarisOvlDepth`, `XSolarisOvlMinColors`, `XSolarisOvlMinRed`, `XSolarisOvlMinGreen`, and `XSolarisOvlMinBlue`. Typically only some subset of these should be specified.
- `XSolarisOvlDepth` specifies that the depth of the selected visual is to be equal to `depth`.
- `XSolarisOvlMinColors` specifies that the selected visual is to have at least `minColors` number of total displayable colors.
- `XSolarisOvlMinRed`, `XSolarisOvlMinGreen`, and `XSolarisOvlMinBlue` can be used to indicate more specific color requirements for `DirectColor` or `TrueColor` visuals. Their corresponding values are specified in `minRed`, `minGreen`, and `minBlue`, respectively. These indicate that the selected visual must have at least the specified number of reds, greens, and/or blues.
- `XSolarisOvlMinBitsPerRGB` specifies that the selected visual is to have at least `minBitsPerRGB` of color channel output from colormaps created on that visual.
- `XSolarisOvlMinBuffers` specifies that the client wants the selected visual to be able to be assigned at least `minBuffers` number of accelerated MBX image buffers.

- `XSolarisOvlUnsharedPixels` selects partner visuals whose window pixels don't lie in the same drawing plane groups as the window pixels of the argument visual `vid`. If a visual uses the same drawing plane group as the argument visual, it is not matched by this criterion.
- `XSolarisOvlUnsharedColors` selects partner visuals whose window pixel colors can be displayed simultaneously when the overlay/underlay window pair has the colormap focus. If a visual shares the same color LUT pool and that pool has only one color LUT in it as the argument visual, the visual is not matched by this criterion.

If either `hardCriteriaMask` of a criteria set is to 0, any visual will match that criteria set with a hard match. Likewise, setting the `softCriteriaMask` of a criteria set to `0`, is sufficient to guarantee at least a soft match for that criteria set.

Selecting an Optimal Overlay/Underlay Visual Pair

The `XSolarisOvlSelectPair` routine is similar to `XSolarisOvlSelectPartner`. However, instead of selecting a partner visual given another visual, this routine simultaneously selects both the overlay and underlay visual from the set of all visual pairs for the given screen. The pair selected is the one that best matches the given criteria. The client is assured that, short of X errors not related to overlays, it can successfully create windows with the returned visuals.

This routine searches through all optimal visual pairs for a given screen, and then through all pairs of visuals (optimal and non-optimal), applying the specified criteria. These criteria are specified in `pCriteria`. Each element of `pCriteria` specifies criteria for both the overlay and underlay. It returns a success or failure status depending on whether it finds a pair that meets all the given criteria.

The selected pair has an overlay that satisfies all the hard criteria specified for the overlay. The pair has an underlay visual that satisfies all the hard criteria for the underlay. The attributes of the overlay visual are returned in `ovVisinfoReturn`. Likewise, the attributes of the underlay visual are specified in `unVisinfoReturn`. If two or more pairs are found that meet all of the hard criteria (both overlay and underlay) and the same number of soft criteria (either overlay or underlay), one of them will be chosen and returned. Which pair is chosen depends on the implementation.

The syntax and arguments are shown below.

```
XSolarisOvlSelectStatus
XSolarisOvlSelectPair (Display *display, int screen, int
numCriteria,
       XSolarisOvlPairCriteria *pCriteria,
       XVisualInfo *ovVisinfoReturn, XVisualInfo *unVisinfoReturn,
       unsigned long *unmetOvCriteriaReturn,
       unsigned long *unmetUnCriteriaReturn)
```

`display`	Specifies the connection to the X server.
`screen`	An integer specifying the screen on which the visuals are to be searched.
`numCriteria`	The number of `XSolarisOvlPairCriteria` structures in the `pCriteria` array.
`pCriteria`	An array of pair criteria structures in priority order from high to low specifying the criteria to be used in selecting the pair.
`ovVisinfoReturn`	A pointer to a caller-provided `XVisualInfo` structure. On successful return, this structure contains a description of the chosen overlay visual.
`unVisinfoReturn`	A pointer to a caller-provided `XVisualInfo` structure. On successful return, this structure contains a description of the chosen underlay visual.
`unmetOvCriteriaReturn`	A pointer to a bitmask that describes the criteria that were not satisfied for the overlay visual. This return argument is meaningful only when the routine returns a value of `XSolarisOvlQualifiedSuccess`, or `XSolarisOvlCriteriaFailure`.

`unmetUnCriteriaReturn` A pointer to a bitmask that describes the criteria that were not satisfied for the underlay visual. This return argument is meaningful only when the routine returns a value of `XSolarisOvlQualifiedSuccess`, or `XSolarisOvlCriteriaFailure`.

Argument Types

`XSolarisOvlPairCriteria` is a structure defining various criteria to be used during visual selection, along with indications of the stringency of the criteria. This structure is defined as:

```
typedef struct {
    XSolarisOvlVisualCriteriaoverlayCriteria;
    XSolarisOvlVisualCriteriaunderlayCriteria;
} XSolarisOvlPairCriteria;
```

`XSolarisOvlVisualCriteria` is defined in the specification of `XSolarisOvlSelectPartner`.

Return Types

Refer to the specification of `XSolarisOvlSelectPartner` for the definition of the type `XSolarisOvlSelectStatus`.

- `XSolarisOvlSuccess` is returned if the search is completely successful in finding a pair that meets all hard and soft criteria of one of the `XSolarisOvlPairCriteria` structures.
- `XSolarisOvlQualifiedSuccess` is returned if the chosen pair satisfies all hard criteria of one of the `XSolarisOvlPairCriteria` structures, but doesn't meet all soft criteria. In this case, `unmetOvCriteriaReturn` and `unmetUnCriteriaReturn` contain the logical OR of the soft criteria that were not met for the overlay and underlay, respectively.
- `XSolarisOvlCriteriaFailure` indicates that no pair could be found that meets all the hard criteria of any of the `XSolarisOvlPairCriteria` structures. In this case, `unmetOvCriteriaReturn` and

`unmetUnCriteriaReturn` contain the logical OR of the hard criteria that were not met by the `XSolarisOvlPairCriteria` structure with the fewest hard failures, for the overlay and underlay, respectively.

- `XSolarisOvlFailure` is returned if some other error is encountered besides criteria match failure.

Criteria Sets

Like `XSolarisOvlSelectPartner`, `XSolarisOvlSelectPair` supports a *degradation sequence* of criteria sets. This means that multiple criteria sets can be specified in a single call. First, the routine attempts to find a pair matching the first criteria set for both the overlay and the underlay. If it finds a pair that meets all of the hard criteria of the first set, it chooses this pair. If no pair meets all hard criteria of the first set, the routine searchs using the second criteria set. This process continues until either a pair is found that meets all of the hard criteria of some criteria set, or all sets have been used to search. This degradation sequence allows clients to specify the criteria for the most preferred pair as the first criteria set. Pairs that are acceptable but less desirable can be specified in criteria sets following the first criteria set. This allows the search to proceed through a progressive relaxation in the client's requirements for the pair with a single subroutine call.

The criteria masks that can be specified are described in "Selecting a Visual for an Overlay/Underlay Window" on page 83.

Security Issues 6

The Solaris environment supports two access control mechanisms: user-based and host-based. It also supports two authorization protocols: MIT-MAGIC-COOKIE-1 and SUN-DES-1. This chapter discusses these access control mechanisms and authorization protocols. It also discusses how to change the server's access control, and how to run clients remotely, or locally as a different user.

Notes About This Chapter

If you run applications in any of the following configurations, you need to read this chapter:

- Linked with a version of `Xlib` *previous* to OpenWindows Version 2 or X11R4. See "Host-Based" on page 94 for details.
- *Statically* linked to OpenWindows Version 2 libraries *and* you want to use the SUN-DES-1 authorization protocol. See "SUN-DES-1" on page 95 for details.
- On a remote server. See "Running Clients Remotely, or Locally as Another User" on page 100 for details.

If you are not using any of the configurations listed above, you do not need to change the default security setup.

6

Access Control Mechanisms

An access control mechanism controls which clients or applications have access to the OpenWindows server. Only properly authorized clients can connect to the server. All unauthorized X clients terminate with the following error message:

```
Xlib: connection to hostname refused by server
Xlib: Client is not authorized to connect to server
```

The server console displays the following message:

```
AUDIT: <Date Time Year>: X: client 6 rejected from IP 129.144.152.193 port 3485
    Auth name: MIT-MAGIC-COOKIE-1
```

The two types of access control mechanisms are: *user-based* and *host-based*. Unless the `-noauth` option is used with `openwin`, both the user-based access control mechanism and the host-based access control mechanism are active. See “Manipulating Access to the Server” on page 97 for more information.

User-Based

A user-based, or authorization-based mechanism allows you to give access explicitly to a particular user on any host. The user’s client passes authorization data to the server. If the data matches the server’s authorization data, the user obtains access.

Host-Based

A host-based mechanism is a general purpose mechanism. It allows you to give access to a particular host, such that all users on that host can connect to the server. This is a weak form of access control; if a host has access to the server, all users on that host can connect to the server.

The Solaris environment provides the host-based mechanism for backward compatibility. Applications linked with a version of `Xlib` older than OpenWindows Version 2 or X11R4 do not recognize the new user-based access

control mechanism. To enable these applications to connect to the server, a user must either switch to the host-based mechanism, or relink with the newer version of `Xlib`.

Note – If possible, clients linked with an older version of Xlib should be relinked with a newer version of Xlib. This enables them to connect to the server with the new user-based access control mechanism.

Authorization Protocols

The OpenWindows environment supports two different authorization protocols: MIT-MAGIC-COOKIE-1 and SUN-DES-1. While they differ in the authorization data used, they are similar in the access control mechanism used.

The MIT-MAGIC-COOKIE-1 protocol, using the user-based mechanism, is the OpenWindows environment default.

MIT-MAGIC-COOKIE-1

The MIT-MAGIC-COOKIE-1 authorization protocol was developed by the Massachusetts Institute of Technology (MIT). A *magic cookie* is a long, randomly generated binary password. At server startup, the magic cookie is created for the server and the user who started the system. On every connection attempt, the user's client sends the magic cookie to the server as part of the connection packet. This magic cookie is compared with the server's magic cookie. The connection is allowed if the magic cookies match, or denied if they do not match.

SUN-DES-1

The SUN-DES-1 authorization protocol was developed by Sun Microsystems. It is based on Secure Remote Procedure Call (RPC) and requires Data Encryption Software (DES) support. The authorization data is the machine-independent netname, or network name, of a user. This data is encrypted and sent to the server as part of the connection packet. The server decrypts the data, and, if the netname is known, allows the connection.

The SUN-DES-1 authorization protocol provides a higher level of security than the MIT-MAGIC-COOKIE-1 protocol. There is no way for another user to use your machine-independent netname to access a server, but it is possible for another user to use the magic cookie to access a server.

This protocol is available only in libraries in the OpenWindows Version 3 and later environments. Any applications built with static libraries, in particular Xlib, in environments prior to OpenWindows Version 3 cannot use this authorization protocol.

"Allowing Access When Using SUN-DES-1" on page 99 describes how to allow another user access to your server by adding their netname to your server's access list.

Changing the Default Authorization Protocol

The default authorization protocol, MIT-MAGIC-COOKIE-1, can be changed to another supported authorization protocol or to no user-based access mechanism at all. The default is changed by supplying options with the `openwin` command. See the `openwin(1)` man page for more information.

For example, to change the default from MIT-MAGIC-COOKIE-1 to SUN-DES-1, start the OpenWindows environment as follows:

```
example% openwin -auth sun-des
```

If you must run OpenWindows without the user-based access mechanism, use the `-noauth` command line option.

```
example% openwin -noauth
```

Warning – Using `-noauth` weakens security. It is equivalent to running OpenWindows with only the host-based access control mechanism; the server inactivates the user-based access control mechanism. Anyone who can run applications on your local machine will be allowed access to your server.

Manipulating Access to the Server

Unless the `-noauth` option is used with `openwin` (see “Changing the Default Authorization Protocol” on page 96), both the user-based access control mechanism and the host-based access control mechanism are active. The server first checks the user-based mechanism, then the host-based mechanism. The default security configuration uses MIT-MAGIC-COOKIE-1 as the user-based mechanism, and an empty list for the host-based mechanism. Since the host-based list is empty, only the user-based mechanism is effectively active. Using the `-noauth` option instructs the server to inactivate the user-based access control mechanism and initializes the host-based list by adding the local host.

You can use either of two programs to change a server’s access control mechanism: `xhost` and `xauth`. For more information, see the `man` pages under `xhost` and `xauth`.. These programs access two binary files created by the authorization protocol. These files contain session-specific authorization data. One file is for server internal use only. The other file is located in the user’s `$HOME` directory:

`.Xauthority` (Client Authority File)

Use the `xhost` program to change the host-based access list in the server. You can add hosts to, or delete hosts from the access list. If you start with the default configuration–an empty host-based access list–and use `xhost` to add a machine name, you lower the level of security. The server allows access to the host you added, as well as to any user specifying the default authorization protocol. See “Host-Based” on page 94 for an explanation of why the host-based access control mechanism is considered a lower level of security.

The `xauth` program accesses the authorization protocol data in the `.Xauthority` client file. You can extract this data from your `.Xauthority` file so that other users can merge the data into their `.Xauthority` file, thus allowing them access to your server, or to the server to which you connect.

See “Allowing Access When Using MIT-MAGIC-COOKIE-1” on page 99 for examples of how to use `xhost` and `xauth`.

Client Authority File

The client authority file is `.Xauthority`. It contains entries of the form:

connection-protocol *auth-protocol* *auth-data*

By default, `.Xauthority` contains MIT-MAGIC-COOKIE-1 as the *auth-protocol*, and entries for the local display only as the *connection-protocol* and *auth-data*. For example, on host *anyhost*, the `.Xauthority` file may contain the following entries:

anyhost:0	MIT-MAGIC-COOKIE-1	*82744f2c4850b03fce7ae47176e75*
localhost:0	MIT-MAGIC-COOKIE-1	*82744f2c4850b03fce7ae47176e75*
anyhost/unix:0	MIT-MAGIC-COOKIE-1	*82744f2c4850b03fce7ae47176e75*

When the client starts up, an entry corresponding to the *connection-protocol* is read from `.Xauthority`, and the *auth-protocol* and *auth-data* are sent to the server as part of the connection packet. In the default configuration, `xhost` returns an empty host-based access list and states that the authorization is enabled.

If you have changed the authorization protocol from the default to SUN-DES-1, the entries in `.Xauthority` contain SUN-DES-1 as the *auth-protocol* and the netname of the user as the *auth-data*. The netname is in the following form:

unix.*userid@NISdomainname*

For example, on host, *anyhost* the `.Xauthority` file may contain the following entries:

anyhost:0	SUN-DES-1	“unix.*15339@EBB.Eng.Sun.COM*”
localhost:0	SUN-DES-1	“unix.*15339@EBB.Eng.Sun.COM*”
anyhost/unix:0	SUN-DES-1	“unix.*15339@EBB.Eng.Sun.COM*”

where unix.*15339@EBB.Eng.Sun.COM* is the machine-independent netname of the user.

Note – If you do not know your network name, or machine-independent netname, ask your system administrator.

Allowing Access When Using MIT-MAGIC-COOKIE-1

If you are using the MIT-MAGIC-COOKIE-1 authorization protocol, follow these steps to allow another user access to your server.

1. **On the machine running the server, use `xauth` to extract an entry corresponding to *hostname:0* into a file.**
   For this example, *hostname* is *anyhost* and the file is *xauth.info*.

```
myhost% $OPENWINHOME/bin/xauth nextract - anyhost:0 > $HOME/xauth.info
```

2. **Send the file containing the entry to the user requesting access (using Mail Tool, `rcp`, or some other file transfer protocol).**

Note – Mailing the file containing your authorization information is a safer method than using `rcp`. If you do use `rcp`, do *not* place the file in a directory that is easily accessible by another user.

3. **The other user must merge the entry into their `.Xauthority` file.**
   For this example, *userhost* merges `xauth.info` into their `.Xauthority` file.

```
userhost% $OPENWINHOME/bin/xauth nmerge - < xauth.info
```

Note – The *auth-data* is session-specific; therefore, it is valid only as long as the server is not restarted.

Allowing Access When Using SUN-DES-1

If you are using the SUN-DES-1 authorization protocol, follow these steps to allow another user access to your server.

1. **On the machine running the server, use `xhost` to make the new user known to the server.**
   For example, to allow new user *somebody* to run on *myhost*, type:

```
myhost% xhost + somebody@
```

2. **The new user must use** `xauth` **to add the entry into their** `.Xauthority` **file.**
   For this example, the new user *somebody*'s machine-independent netname is unix.*15339@EBB.Eng.Sun.COM*.

```
userhost% echo 'add myhost:0 SUN-DES-1 "unix.15339@EBB.Eng.Sun.COM"' | $OPENWINHOME/bin/xauth
```

Running Clients Remotely, or Locally as Another User

X clients use the value of the `DISPLAY` environment variable to get the name of the server to which they should connect.

To run clients remotely, or locally as another user, follow these steps:

1. **On the machine running the server, allow another user access.**
   Depending on which authorization protocol you use, follow the steps outlined in either "Allowing Access When Using MIT-MAGIC-COOKIE-1" on page 99 or "Allowing Access When Using SUN-DES-1" on page 99.

2. **Set** `DISPLAY` **to the name of the host running the server.**
   For this example, the host is *remotehost*.

```
myhost% setenv DISPLAY remotehost:0
```

3. **Run the client program.**
   The client is displayed on the remote machine, *remotehost*.

```
myhost% client_program&
```

Reference Display Devices A

This appendix presents information on the Solaris reference display devices and the visuals they export. For more information on visuals, see Chapter 3, “Visuals on the Solaris X Server.”

Solaris Reference Display Devices

Certain display devices are considered to be *reference devices* in the Solaris environment. These devices have example device handlers provided in the Solaris Driver Developer Kit (DDK). You can use the reference device handler example code as a template for your own device handler.

The process of writing and configuring a device handler is described in the *X Server Device Developer’s Guide*, which is included in the Solaris DDK product. The Solaris X server supports any device for which a valid device handler is written and configured into the system.

Solaris Reference Devices and Visuals

Table A-1 lists the reference display devices and the visuals that they export. The device name specifies the display adapter to the server, and the product name specifies the type of display card. Note that if there is a distinct product name for a device, the product name is used in preference to the CG*n* device name (for example, TC is used, not CG8).

Exported depths specify the depths of the visuals advertised by the server for screens of this particular device type. MPG (Multiple Plane Group) indicates that the device supports multiple depth visuals. For other information on terms used in this table, see “Glossary” on page 127.

Table A-1 Solaris Reference Display Devices

Device Name	Product Name	Device Driver	Bus	Exported Depths
BW2	None	`/dev/fbs/bwtwo`*X*	SBus, VME/obio, P4	1-bit
CG3	None	`/dev/fbs/cgthree`*X*	SBus	8-bit
CG6	GX	`/dev/fbs/cgsix`*X*	SBus, P4	8-bit
CG6	GXplus/ TurboGXplus	`/dev/fbs/cgsix`*X*	SBus	8-bit
CG8	TC	`/dev/fbs/cgeight`*X*	SBus, P4	1, 24-bit (MPG)
vga4	VGA	Not applicable	ISA, EISA, MCA	8-bit
vga8	VGA	Not applicable	ISA, EISA, MCA	8-bit
i8514	8514/A	Not applicable	ISA, EISA, MCAS	8-bit

Note – The server is configured to support a maximum of 16 displays; any limitations you might encounter are the number of frame buffers your hardware supports.

SPARC: Supported Reference Devices

BW2

The BW2 is a simple 1-bit frame buffer supporting monochrome monitors. The device handler for this device exports the 1-bit StaticGray visual only. Therefore, this is the built-in default visual. A variety of BW2 frame buffers are available for different buses and screen resolutions, including third-party offerings.

CG3

The CG3 is a simple 8-bit indexed color, dumb frame buffer for SBus systems. The device handler for this device exports several 8-bit visuals (listed in the following sections). The built-in default visual is 8-bit PseudoColor.

GX Family of Devices

The GX is an 8-bit indexed color graphics accelerator, specializing in 2D and 3D wireframe, flat-shaded polygon, and general window system acceleration. Window system acceleration is automatic; you can access other acceleration features through Solaris graphics APIs. Several 8-bit visuals are supported, and the built-in default visual is 8-bit PseudoColor. The GX is available for SBus and P4 bus.

The GXplus device is similar to the GX with additional memory that can be used for double buffering and expanded screen resolution on SBus systems. The Solaris X server uses the GXplus to automatically accelerate X11 pixmaps by using offscreen storage whenever possible.

TC (CG8)

The TC device possesses two separate memory buffers, or *plane groups*: 1-bit monochrome and 24-bit color. Windows may be created in both plane groups; therefore, it is an MPG device. All 1-bit and 24-bit visuals are supported.

Some (older) X11 client applications assume that color frame buffers use an 8-bit built-in default visual and do not run in color on the TC. To avoid this, the built-in default visual is 1-bit StaticGray.

The plane groups of the TC do not conflict with each other; they are completely separate memory buffers. OpenWindows takes advantage of this to increase system performance by not damaging 1-bit windows when they are occluded by 24-bit windows, and vice versa. This behavior is called *minimized exposure*. Use the `-nominexp` option of `openwin(1)` to disable this behavior. If this option is used, 1-bit windows will damage 24-bit windows and 24-bit windows may damage 1-bit windows.

The Solaris X server also provides minimized exposure for other MPG devices, when applicable. Use the `-nominexp` option of `openwin` with these devices.

Note – The X protocol states that cursor components can be arbitrarily transformed. To enhance general system performance, the OpenWindows server always renders the cursor in the 1-bit plane group of the TC.

x86: Supported Reference Devices

VGA

The VGA is a simple color dumb frame buffer. The server supports VGA as 8-bit indexed color with all visual types and a default of PseudoColor (vga8), or 4-bit StaticColor (vga4). When using 8-bit mode, the resolution is most often 1024x768. Four-bit mode is often limited to a resolution of 640x480 because this is the basic VGA graphics mode that is available on all VGA devices. Most VGAs provide a bitsPerRGB of 6. The vga8 server is also capable of supporting the XGA as a dumb frame buffer.

Support for VGA panning is available in modes of the 4-bit VGA. Panning mode provides the ability to have a physical window that maps onto a larger virtual display. Movement within the virtual display is performed by “pushing” the mouse past the edge of the screen. The display automatically moves the physical window in the virtual display in the direction that the mouse was pushed until the physical window touches the edge of the virtual boundary.

Use panning only if you are an experienced OpenWindows user. Icons and pop-up boxes (menus, dialogs, and so on) can appear off screen with no immediate visible notification. You must be experienced enough to recognize these situations, and be able to recover by looking for the hidden window objects. Pop-up pointer jumping is highly recommended while using panning. Virtual window managers, such as `olvwm` or `tvwm`, can cause additional confusion; do not use them.

8514/A

The 8514/A is an 8-bit indexed color graphics accelerator providing general window system acceleration. This device provides substantially improved performance compared to a VGA. The server limits its support of 8514/A to

8-bit indexed color and a resolution of 1024x768 or 1280x1024. It supports all 8-bit visuals. The built-in visual is 8-bit PseudoColor. Most 8514/A accelerators provide a bitsPerRGB of 6.

A

Multi-Buffering Application Program Interface, Version 3.2

B

This appendix describes the C-language application program interface (API) to the Multi–Buffering (MBX) extension.[1] These routines provide direct access to the protocol and add no semantics.

This appendix assumes that you are familiar with the MBX protocol described in the MIT standard, *Extending X for Double-Buffering and Multi-Buffering, and Stereo, Version 3.*2. See "X Consortium Extensions" on page 4 for information on how to access this standard.

Throughout this appendix, the file path names given are relative to `/usr/openwin`.

Library File

These API routines can be accessed by dynamically linking with the shared object file, `lib/libXext.so`.

Note – Although a statically linkable version of this same library, `libXext.a`, is available in the same directory, static linking is not recommended because this reduces application compatibility with future releases.

---

1. This document is derived from the document *Multi-Buffering Application Program Interface* by David P. Wiggins (dwig@sr71.b11.ingr.com), Intergraph Corporation, Version 1.0.

B

Header File

The header file for this extension is `include/X11/extensions/multibuf.h`. This file defines the following types, constants, structures, and functions.

New Routines

The following routines are added by this API:

- `XmbufQueryExtension`
- `XmbufGetVersion`
- `XmbufCreateBuffers`
- `XmbufDestroyBuffers`
- `XmbufDisplayBuffers`
- `XmbufGetWindowAttributes`
- `XmbufChangeWindowAttributes`
- `XmbufGetBufferAttributes`
- `XmbufChangeBufferAttributes`
- `XmbufGetScreenInfo`

Note – `XmbufCreateStereoWindow` is not supported in SunSoft's MBX implementation.

New Types

Buffer identifiers are held in a new drawable type, `Multibuffer`. A `Multibuffer` can be substituted in all X calls where a `Drawable` is specified.

New Constants

The following constants are defined in the `multibuf.h` header file.

Event Type Constants

- `MultibufferClobberNotify`
- `MultibufferUpdateNotify`

Error Constants

- `MultibufferBadBuffer`

Update Action Constants

- `MultibufferUpdateActionUndefined`
- `MultibufferUpdateActionBackground`
- `MultibufferUpdateActionUntouched`
- `MultibufferUpdateActionCopied`

Update Hint Constants

- `MultibufferUpdateHintFrequent`
- `MultibufferUpdateHintIntermittent`
- `MultibufferUpdateHintStatic`

Window Mode Constants

- `MultibufferModeMono`

Note – SunSoft's MBX implementation does not support the window mode constant `MultibufferModeStereo`, and the window side constants `MultibufferSideMono`, `MultibufferSideLeft`, and `MultibufferSideRight`

Event Mask Constants

- `MultibufferClobberNotifyMask`
- `MultibufferUpdateNotifyMask`

Valuemask Constants

- `MultibufferWindowUpdateHint`
- `MultibufferBufferEventMask`

B

Clobber State Constants

- `MultibufferUnclobbered`
- `MultibufferPartiallyClobbered`
- `MultibufferFullyClobbered`

New Structures

Several new structure types are defined. Most are introduced in the function discussion of the function that requires a structure as a parameter. The following structures are not parameters in any of the functions discussed in “MBX Functions” on page 111.

MultibufferClobberNotify Event

```
typedef struct {
    int type;      /* = mbuf_event_base + MultibufferClobberNotify */
    unsigned long serial; /* # of last request processed by server */
    int send_event;    /* true if this came from a SendEvent request */
    Display *display;     /* Display the event was read from */
    Multibuffer buffer;   /* buffer of event */
    int state;            /* see Clobber state constants above */
} XmbufClobberNotifyEvent;
```

MultibufferUpdateNotify Event

```
typedef struct {
    int type;        /* = mbuf_event_base + MultibufferUpdateNotify */
    unsigned long serial; /* # of last request processed by server */
    int send_event; /* true if this came from a SendEvent request */
    Display *display;     /* Display the event was read from */
    Multibuffer buffer;   /* buffer of event */
} XmbufUpdateNotifyEvent;
```

MBX Functions

The following functions generate MBX protocol requests. Except for `XmbufQueryExtension`, if any of them are called with a display that does not support the MBX extension, the `ExtensionErrorHandler` (registered by `XSetExtensionErrorHandler`) is called. If the `ExtensionErrorHandler` returns (does not exit the program), most of the MBX functions return an error.

XmbufQueryExtension

This function determines whether a display supports the MBX extension.

```
Bool
XmbufQueryExtension(display,mbuf_event_base,mbuf_error_base)
Display *display;
int *mbuf_event_base; /* RETURN */
int *mbuf_error_base; /* RETURN */
```

Arguments

display
Specifies the connection to the X server.

mbuf_event_base
Returns the first event code used by the extension. An `XEvent` with a type field equal to **mbuf_event_base* + `MultibufferClobberNotify` is a `ClobberNotify` event. An `XEvent` with a type field equal to **mbuf_event_base* + `MultibufferUpdateNotify` is an `UpdateNotify` event.

mbuf_error_base
Returns the first error code used by the extension. An `XErrorEvent` with an `error_code` field equal to `*mbuf_error_base` + `MultibufferBadBuffer` is a `BadBuffer` error.

Description

If the given display supports the MBX extension, `XmbufQueryExtension` fills in `*mbuf_event_base` and `*mbuf_error_base` and returns `True`, else it returns `False` without changing `*mbuf_event_base` and `*mbuf_error_base`.

XmbufGetVersion

This function retrieves the major and minor version numbers of the MBX extension.

```
Status
XmbufGetVersion(display,major_version,minor_version)
Display *display;
int *major_version; /* RETURN */
int *minor_version; /* RETURN */
```

Arguments

display
Specifies the connection to the X server.

major_version
Returns the major version number of the extension.

minor_version
Returns the minor version number of the extension.

Description

If no error occurs, `XmbufGetVersion` fills in `*major_version` and `*minor_version` with the version of the extension supported by the display and returns non–zero, else it returns zero without changing `*major_version` and `*minor_version`.

Protocol

Issues a `GetBufferVersion` request.

XmbufCreateBuffers

This function requests a specified number of image buffers to be associated with a window.

```
int
XmbufCreateBuffers(display,window,count,update_action,
                   update_hint,buffers)
Display *display;
Window window;
int count;
int update_action, update_hint;
Multibuffer *buffers; /* RETURN */
```

Arguments

display
Specifies the connection to the X server.

window
Specifies the window with which the buffers should be associated.

count
Specifies the number of buffers.

update_action
Specifies the update action to be applied to the buffers. See “Update Action Constants” on page 109 for allowable values.

update_hint
Specifies the update hint for the buffers. See “Update Hint Constants” on page 109 for allowable values.

buffers
Must be a pointer to enough memory to hold *count* multibuffers. Returns the multibuffer IDs that were created.

Description

`XmbufCreateBuffers` attempts to create `count` buffers associated with the given window. The requested number of buffers may not be able to be satisfied and less than *count* buffers may actually be allocated. The number of buffers actually allocated is returned. This many multibuffer IDs will be returned in `*buffers`. If an error occurs, `XmbufCreateBuffers` returns zero and leaves `*buffers` undefined.

`buffers` must always be large enough to hold at least `count` multibuffers.

The buffers are assigned the given `update_action` and `update_hint`.

No `BadAlloc` errors are ever generated due to lack of buffers because, in the worst case, `buffers[0]` can always be associated with the existing displayed image buffer of the window. In this case, one buffer still can be returned. However, `BadAlloc` may still be returned if temporary memory needed to execute the request cannot be allocated.

Diagnostics

BadWindow
`window` does not name a defined window.

BadValue
`update_action` or `update_hint` is invalid.

BadIDChoice
At least one of the multibuffer IDs in `buffers` is an invalid resource ID.

BadMatch
`window` is an `InputOnly` window.

BadAlloc
The system failed to allocate the necessary temporary memory to execute the request.

Protocol

Issues a `CreateImageBuffers` request.

XmbufDestroyBuffers

This function frees the window's associated image buffers.

```
void
XmbufDestroyBuffers(display,window)
Display *display;
Window window;
```

Arguments

display
Specifies the connection to the X server.

window
Specifies the window whose buffers are to be destroyed.

Description

Destroys the image buffers associated with the `window`.

Diagnostics

BadWindow
`window` does not name a defined `Window`.

Protocol

Issues a `DestroyImageBuffers` request.

XmbufDisplayBuffers

This function tells the system which image buffers are visible in the given windows.

```
void
XmbufDisplayBuffers(display,count,buffers,min_delay,max_delay)
Display *display;
int count;
Multibuffer *buffers;
int min_delay, max_delay;
```

Arguments

display
Specifies the connection to the X server.

count
Specifies the number of multibuffer IDs pointed to by buffers.

buffers
Specifies the `Multibuffers` selected for display in their associated windows.

min_delay
Specifies the minimum number of milliseconds that must elapse since the last time a `DisplayImageBuffers` was executed on a window.

max_delay
Specifies an additional delay beyond `min_delay` that the server is allowed to wait to complete the `DisplayImageBuffers` request.

Description

If no error occurs, `XmbufDisplayBuffers` displays the indicated buffers in their associated windows within the given time constraints.

Diagnostics

BadBuffer
At least one of the `Multibuffers` in `buffers` does not name a defined Buffer.

BadMatch
Two or more `Multibuffers` associated with the same window were specified in `buffers`.

BadAlloc
The system failed to allocate the necessary temporary memory to execute the request.

Protocol

Issues a `DisplayImageBuffers` request.

XmbufGetWindowAttributes

This function retrieves a window's multi-buffering attribute values.

```
Status
XmbufGetWindowAttributes(display,window,attributes)
Display *display;
Window window;
XmbufWindowAttributes *attributes; /* RETURN */
```

Arguments

display
Specifies the connection to the X server.

window
Specifies the window whose multibuffer attributes are to be retrieved.

attributes
Returns the specified window's multibuffer attributes.

Description

If no error occurs, `XmbufGetWindowAttributes` returns non–zero and stores the window's multibuffer attributes in the `XmbufWindowAttributes` structure. To free the buffers list in the attributes structure, use `XFree`. If an error occurs, `XmbufGetWindowAttributes` returns zero and leaves *attributes* unchanged.

Structures

```
typedef struct {
    int displayed_index;/* which buffer is being displayed */
    int update_action;/* see Update action constants */
    int update_hint;/* see Update hint constants */
    int window_mode;/* see Window mode constants */
    int nbuffers;/* number of buffers in following list */
    Multibuffer *buffers;/* buffer IDs associated with this window */
} XmbufWindowAttributes;
```

Diagnostics

BadWindow
`window` does not name a defined `Window`.

BadAccess
`window` is not multi–buffered.

BadValue
Not currently generated.

BadAlloc
The system failed to allocate the necessary temporary memory to execute the request.

Protocol

Issues a `GetMultiBufferAttributes` request.

XmbufChangeWindowAttributes

This function modifies a window's multi-buffering attribute values.

```
void
XmbufChangeWindowAttributes(display,window,valuemask,values)
Display *display;
Window window;
unsigned long valuemask;
XmbufSetWindowAttributes *values;
```

Arguments

display
Specifies the connection to the X server.

window
Specifies the window whose multibuffer attributes are to be changed.

valuemask
Specifies which attributes are to be changed using information in the specified attributes structure. The only value currently defined for this is `MultibufferWindowUpdateHint`.

values
Specifies any values as indicated by `valuemask`.

Description

If no error occurs, `XmbufChangeWindowAttributes` sets the multi-buffering attributes that apply to all buffers associated with the given window.

Structures

```
typedef struct {
    int update_hint;      /* see Update hint constants above */
} XmbufSetWindowAttributes;
```

Diagnostics

BadWindow
`window` does not name a defined `Window`.

BadMatch
`window` is not multi–buffered.

BadValue
`update_hint` or `valuemask` is invalid.

Protocol

Issues a `SetMultiBufferAttributes` request.

XmbufGetBufferAttributes

This function retrieves an individual image buffer's attributes.

```
Status
XmbufGetBufferAttributes(display,buffer,attributes)
Display *display;
Multibuffer buffer;
XmbufBufferAttributes *attributes; /* RETURN */
```

Arguments

display
Specifies the connection to the X server.

buffer
Specifies the buffer whose attributes are to be retrieved.

attributes
Returns the per–buffer attributes for the specified buffer.

Descriptions

If no error occurs, `XmbufGetBufferAttributes` fills in the `attributes` structure with values of the per–buffer attributes for the indicated buffer and returns non–zero, else it returns zero and leaves `attributes` unchanged.

Structures

```
typedef struct {
    Window window;     /* which window this buffer belongs to */
    unsigned long event_mask;/* events selected for this buffer */
    int buffer_index;      /* which buffer is this */
    int side;              /* see Window side constants above */
} XmbufBufferAttributes;
```

Diagnostics

BadBuffer
`buffer` does not name a defined `Buffer`.

BadValue
Not currently generated.

Protocol

Issues a `GetBufferAttributes` request.

XmbufChangeBufferAttributes

This function modifies an individual image buffer's attribute values.

```
void
XmbufChangeBufferAttributes(display,buffer,valuemask,values)
Display *display;
Multibuffer buffer;
unsigned long valuemask;
XmbufSetBufferAttributes *values;
```

Arguments

display
Specifies the connection to the X server.

buffer
Specifies the buffer whose attributes are to be changed.

valuemask
Specifies the specific buffer attributes to be changed. The only value currently defined for this is `MultibufferBufferEventMask`.

values
Specifies any values as indicated by `valuemask`.

Description

If no error occurs, `XmbufChangeBufferAttributes` sets the attributes for the indicated buffer.

Structures

```
typedef struct {
    unsigned long event_mask;/* see Event mask constants above */
} XmbufSetBufferAttributes;
```

Diagnostics

BadBuffer
`buffer` does not name a defined `Buffer`.

BadValue
`valuemask` or `values->event_mask` is invalid.

Protocol

Issues a `SetBufferAttributes` request.

XmbufGetScreenInfo

This function retrieves information about the visuals on a screen that support multi-buffering.

```
Status
XmbufGetScreenInfo(display,drawable,nmono,mono_info,nstereo,
     stereo_info)
Display *display;
Drawable drawable;
int *nmono;                    /* RETURN */
XmbufBufferInfo **mono_info;  /* RETURN */
int *nstereo;                  /* RETURN */
XmbufBufferInfo **stereo_info;/* RETURN */
```

Arguments

display
Specifies the connection to the X server.

drawable
Specifies a drawable on the screen whose buffer information is to be retrieved.

nmono
Returns the number of entries in the `mono_info` list.

mono_info
Returns a list of structures describing which monoscopic visuals are multi–buffered.

nstereo
Returns the number of entries in the `stereo_info` list.

Note – The stereo features of MBX are not supported in Solaris, so the value of `nstereo` is always `0` for all screens.

---

stereo_info
Returns a list of structures describing which stereoscopic visuals are multi–buffered.

Description

If no error occurs, `XmbufGetScreenInfo` returns non–zero and gets the parameters defining the characteristics of the multi-buffered windows that may be created on the screen of the given drawable. If `*nmono` is greater than zero, then `*mono_info` is set to the address of an array of `XmbufBufferInfo` structures describing the various visuals and depths that may be used to create multi-buffered windows. Otherwise, `*mono_info` is set to `NULL`. To release the storage returned in `*mono_info`, use `XFree`. If an error occurs, `XmbufGetScreenInfo` returns zero and leaves `*nmono`, `*mono_info`, `*nstereo`, and `*stereo_info` unchanged.

Structures

```
typedef struct {
    VisualID visualid;    /* visual usable at specified depth */
    int max_buffers;      /* max. num. of bufs for this visual */
    int depth;            /* depth of buffers creatable */
} XmbufBufferInfo;
```

Diagnostics

BadDrawable
`drawable` does not name a defined `Drawable`.

BadAlloc
The system failed to allocate the necessary temporary memory to execute the request.

Protocol

Issues a `GetBufferInfo` request.

B

Glossary

Access Control Mechanism

An access control mechanism is a means of deciding which clients or applications have access to the OpenWindows server. There are two different types of access control mechanisms: user-based and host-based.

Bitmap

A bitmap is a rectangular array of elements, where each element holds either an *inside* value or an *outside* value.

Bitmap Font

A bitmap font is a collection of bitmaps with additional information (for example, character spacing) that defines how the bitmaps are to be used.

Bus

The bus is the system input/output (I/O) link. The display device is both physically and logically connected to the system by the bus. The SBus, VME, and P4 buses are used in SPARC systems. A third-party system may use a bus other than one of these three buses.

Client

A client is an application program that connects to the window server by some interprocess communication. It is referred to as a client of the window server. A client can run on the same machine as the window server or it can connect to a server running on another machine on the network. A client of the OpenWindows server must communicate via the X11 protocol.

Client-Server Model

The most commonly used paradigm when writing distributed applications is the client-server model. In this scheme, clients request services from a window server process. The client and server require a protocol that must be implemented at both ends of a connection. The OpenWindows server implements the X11 protocol.

Color Look-Up Table

A color look-up table is a hardware device that provides a mapping between pixel values and RGB color values. Also called a look-up table (LUT).

Colormap Flashing

Only one client colormap is installed at a given time. The windows that are associated with the installed colormap will show their correct colors. Windows that are associated with some other colormap may show false colors. This display of false colors is referred to as colormap flashing.

Composite Font

A composite font is a collection of base fonts organized hierarchically.

Connection

The communication path between a client and the server.

Default Visual

The default visual is one of the visuals available on the display device. When you start a client program, the program will usually run in the default visual unless a different visual is specified.

Display Device

Your monitor is connected to a display device that controls what is shown on the monitor. The display device includes memory (called a frame buffer) dedicated to storing display information. A display device is also referred to as a graphics adapter.

Device Driver

The device driver is the name of a device in the UNIX file system, where *X* is the number of that particular device on your system. For example, if a system had two CG3s, the first would be named `/dev/fbs/cgthree0`, and the second would be `/dev/fbs/cgthree1`. If a system had one CG3 and one GX, the CG3 would be `/dev/fbs/cgthree0` and the GX `/dev/fbs/cgsix0`.

Event

Clients are informed of information asynchronously by means of events. Events are grouped into types. A client must express interest in an event in order to receive that event from the server.

Extension

An extension to the core protocol can be defined to extend the functionality of the system.

Frame Buffer

Pixel data is typically stored in dedicated computer memory known as a frame buffer or video memory.

Graphics Accelerator

A display device that includes circuitry to increase the rate at which images are drawn into the frame buffer is called an accelerator, or graphics accelerator. A graphics accelerator often includes memory and circuitry that permits enhanced functionality, such as display of additional colors, 3D images, and animation.

Graphics Adapter

See Display Device.

Hardware Colormap

A hardware colormap is a color LUT. (*See also* Color Look-Up Table).

Look-Up Table

See Color Look-Up Table.

Multi-Depth Device

The TC display device provides visuals of different depths; it is referred to as a multiple plane group (MPG) or multi-depth device.

Multiple Plane Group

A display device that can simultaneously support more than one visual category is known as a multiple plane group (MPG) device.

Outline Font

An outline font is a collection of *ideal* shapes of characters. Each shape is defined numerically by continuous curve segments that separate the *inside* from the *outside* of the shape. This method is in use on high-resolution devices such as photo-typesetters.

Pixmap

A pixmap is a block of off-screen memory in the server; it is an array of pixel values.

Plane Group

The physical memory on a display device in which the pixel data is stored is commonly called a plane group.

Product Name

The product name identifies the type of display card.

Request

A request is a command to the server sent over a connection.

RGB

R, G, and B are the voltage levels to drive the red, green, and blue monitor guns, respectively.

Screen

A screen is a physical monitor and hardware, which is either color or black-and-white. A typical configuration could be a single keyboard and mouse shared among the screens.

Software Colormap

A software colormap is a software abstraction of the color mapping process that a color LUT provides. The software colormap can be loaded, or installed, into a hardware color LUT. Also called a colormap.

Virtual Colormap

A software colormap that is not visible until it is installed into a hardware color LUT.

Visual

A visual describes a way of interpreting a pixel value. The visual class and the pixel size attribute collectively describe a visual.

Visual Category

A visual category is a grouping of all visual classes of a given pixel size. The following visual categories are supported by OpenWindows: 1-bit, 4-bit, 8-bit, and 24-bit.

Visual Class

A visual class is how the pixel will be displayed as a color.

Window

A window provides a drawing surface to clients for text and graphics. A single client application can use multiple windows.

Window ID Table Descriptor

A window ID (WID) table contains descriptors for visual aspects of a pixel, such as whether it is an 8-bit pixel or a 24-bit pixel, which LUT should be used when displaying the pixel, and whether the pixel is double-buffered.

Window Manager

Manipulation of windows on the screen and much of the user interface (policy) is typically provided by a window manager client. The window manager communicates only with the window server.

Window Server

A window server, or display server such as the Solaris X server, is a program that handles the display capabilities of a machine and collects input from user devices and other clients, and sends events to clients. The server handles all communication with the window manager.

Index

Symbols

.Xauthority file 97–98

A

Adobe FTP site 23

Adobe public access file server 23

authorization protocols, *See* security

authorization-based access control mechanism, *See* security

B

bdftopcf 46

bitmap distribution format 46

bitmap fonts 46–47, 52

Black pixel location note 14

bus, used in SPARCsystems 127

BW2 display device, description of 102
- *See also* display devices

C

CG3 display device, description of 103
- *See also* display devices

CG6 display device, *See* GX display device and GXplus display device

CG8 display device, *See* TC display device

client
- running locally as another user 100
- running remotely 100

client library
- for DPS 18

color
- color name database 14
- recommendations 14

compose key support 13

compressing font files 47

contexts
- and DPS 18
- secure 21
- three ways to share VM 21

D

DES (Data Encryption Software), with SUN-DES-1 95

display devices
- bus, used in Sun SPARCsystems 127
- BW2

description of 102
CG3
description of 103
CG6, *See* GX and GXplus
CG8, *See* TC
GT
window damage note 103
GX
description of 103
GXplus
description of 103
programming hints 37
supported devices table 102
TC
description of 103
DISPLAY environment variable 100
DPS
.upr files 50
changing the resource path 50
client library 18
font enhancements 20
libraries supported 20
PostScript interpreter 18
pswrap translator 18
security issues 21

F

F3 fonts 47
character set supported by 55
F3BitMap resource 47
font management library, definition of 3
fonts
.afm file 45
.enc file 45, 55
.map file 45
.ps file 45
.trans file 45
.upr file 45, 53
adding bitmap fonts 52
adding new fonts 51–56
adding scalable fonts 53, 56
and X terminals 56
default font path in X11 48
files included in openwindows 45
fonts.alias file 52, 55
fonts.dir file 52
fonts.scale file 54
formats 44–45
outline and bitmap 46, 47
replacing outline with bitmap 47
using F3 fonts 47
ftp program 4
ftp, accessing Adobe FTP site 23

G

GX display device, description of 103
See also display devices
GXplus display device, description of 103
See also display devices

H

host-based access control mechanism, *See* security

L

libraries
DPS. list of 20
X, list of 8

M

makebdf 46
makepsres 53

MBX (multi-buffering) X extension 5
MIT-MAGIC-COOKIE-1 authorization protocol, *See* security
MIT-SHM (Shared Memory) X extension 5
mkfontdir 52
multiple plane group, characteristics of 37

N

NISdomainname, definition of 98

O

openwin command
-noauth option 94, 96
outline fonts 46, 47
overlay windows
advanced features
background 61
backing store 62
border 62
choosing visuals 65–66
colormap 63
gravity 63
input distribution model 63
print capture 64–65
and existing pixel transfer routines 81–82
and existing primitive rendering routines 71–72
basic features
creation 68
definition 59–70
viewability 60

P

portable compiled format 46
compressed files 47
PostScript interpreter 18
pswrap translator 18

R

resource files (.upr) 50
RPC (Remote Procedure Call), with SUN-DES-1 95

S

scalable fonts 53–56
secure context creation 21
security 93–100
.Xauthority file 97–98, 99
contents with MIT-MAGIC-COOKIE-1 98
contents with SUN-DES-1 98
access control mechanisms 94–95
definition of 94
how both are active 97
authorization protocols 95–96
default configuration 95
default, how to change 96
authorization-based, *See* user-based
clients
running locally as another user 100
running remotely 100
connection attempt error message 94
default configuration 95
determining if configuration change is required 93
host-based, backward compatibility 94
host-based, definition of 94
MIT-MAGIC-COOKIE-1 authorization protocol 95
NISdomainname, definition of 98

-noauth option 94
weakens security warning 96
server
manipulating access 97–100
allowing access with MIT-MAGIC-COOKIE-1 99
allowing access with SUN-DES-1 99
SUN-DES-1 authorization protocol
definition of 95–96
need to reconfigure 93
user-based, definition of 94
userid, definition of 98
xauth program 97, 99
xhost program 99
server
applications that run with 8
architecture diagram 3
changing X11 default font path 48–49
DIX layer, definition of 3
font management library, definition of 3
manipulating access control 97–100
OS layer, definition of 3
SHAPE X extension 5
SUN-DES-1 authorization protocol, *See* security
system file access 21

T

TC display device, description of 103
See also display devices

U

user-based access control mechanism, *See* security

V

virtual memory 18
visuals
default
get with XGetVisualInfo function 36
gamma-corrected 38–42
multiple plane group, characteristics of 37
VM (virtual memory) 18, 21
shared 18

W

White pixel location note 14

X

X
applications not supported 10
applications supported 9
compose key 13
extensions
MBX (multi-buffering) 5
MIT-SHM (Shared Memory) 5
SHAPE 5
XInput 5
XTEST 6
libraries supported 8
terminals and fonts 56
X Consortium
extensions supported 4
xauth program 97, 99
XCopyArea, and overlay windows 82
XCopyPlane, and overlay windows 82
XDPSCreateSecureContext 22
XGetImage, and overlay windows 81
XGetVisualInfo function

list default visual 36
xhost program 99
XInput X extension 5
XLFD
font naming convention 54
xlsfonts 52, 55
XOvlSelectPair 88
XSetFontPath 49
XTEST X extension 6