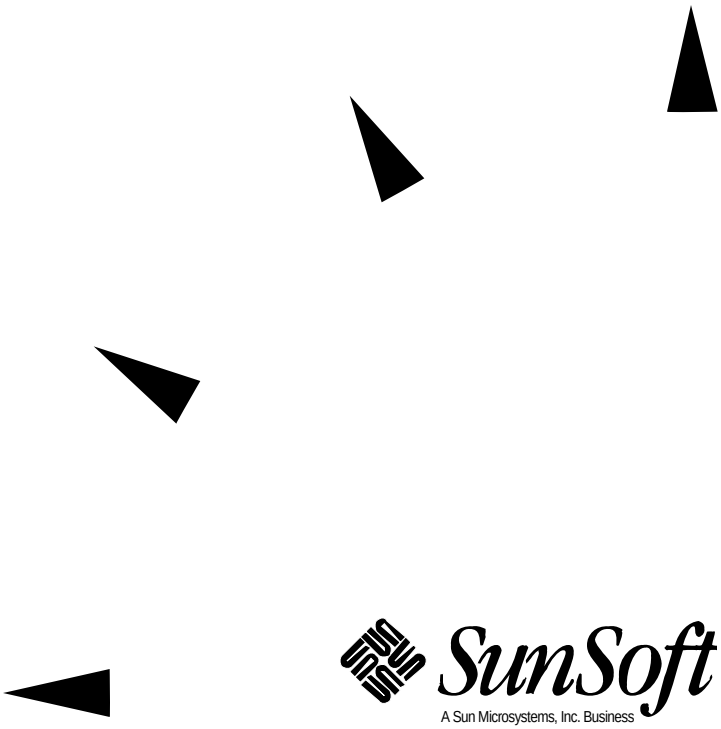
OpenWindows Server Programmer's Guide



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

SunSoft
A Sun Microsystems, Inc. Business

Contents

Preface xv

New Features xxi

1. Overview of OpenWindows Architecture 1

Architecture Overview 1

OPEN LOOK Graphical User Interface 3

OpenWindows Applications 3

OpenWindows Application Programmer's Interfaces 4

OpenWindows Toolkits 4

Libraries 5

OPEN LOOK Window Manager 6

OpenWindows Server 6

OpenWindows Directory Structure 7

2. Introduction to the OpenWindows Server 11

Server Architecture 12

X11R5 Server 12

MIT X Extensions 13
DPS Extension 14
Applications That Run With the Server 14
3. X Features and Enhancements 17
Overview of the X Window System 17
X Protocol 18
The X Library 18
X Toolkits 19
X11 Features 19
X11 Libraries 19
Supported X11 Applications 20
Unsupported Applications 21
ICCCM Compliance 21
MIT X Extensions Supported 21
How To Access MIT X11 Extension Standards 22
XInput Extension 22
MBX (Multi-Buffering) Extension 22
SHAPE Extension 23
MIT-SHM (Shared Memory) Extension 23
XTEST Extension 23
MIT-SUNDRY-NONSTANDARD 23
Notes on X11 Programming 24
Compose Key Support 24
Color Name Database 24

Color Recommendations. 24
4. DPS Features and Enhancements. 27
Introduction to the DPS System 27
How Applications Use the DPS System 28
DPS Extension to X 28
DPS Font Enhancements 30
DPS Libraries 31
Adobe NX Agent Support 31
Applications Modified to use DPS. 31
DPS Security Issues 32
System File Access. 32
Secure Context Creation 32
How to Access Information From Adobe 33
When DPS Encounters Internal Errors 34
5. Font Support 35
Font Formats. 35
Outline and Bitmap Fonts. 36
Replacing Outline Fonts with Bitmap Fonts. 37
When Replacement Occurs. 37
Using F3 Fonts in DPS. 38
Locating Fonts 38
Font Directory Structure 38
Changing the Default Font Path in X11. 39
Changing the Resource Path in DPS 40

Font File Suffixes 42
Adding New Fonts 43
Adding Bitmap Fonts 43
Adding Outline Fonts 44
Using OPEN LOOK Fonts on X Terminals 47
6. Visuals and Display Devices 49
Display Devices 49
Reference Display Devices 49
SPARC Supported Reference Devices 49
x86 Supported Reference Devices 50
IHV Display Devices 50
Visuals 50
Multiple Depth Devices 50
Default Visual. 51
Reference Devices and Visuals 51
Reference Devices 51
SPARC Device-Specific Information 52
x86 Device-Specific Information 54
The Default Visual 55
Changing the Screen Default Visual 55
SPARC Example 56
x86 Example 57
Hints for Windows Programming With Visuals 57
Default Visual Assumptions 57

Multiple Hardware Colormaps 58
Colormap Installation–Multiple LUT Devices 58
Colormap Demo 58
Gamma-Corrected Visuals 59
7. Visual Overlay Windows 65
Basic Features of Overlay Windows 65
Definition 65
Creating an Overlay Window 66
Overlay Window Viewability 67
Rendering Transparency 67
Advanced Features of Overlay Windows 67
Overlay Window Background 67
Overlay Window Border 68
Overlay Window Backing Store 69
Overlay Window Gravity 69
Overlay Colormap 69
Other Overlay Window Characteristics 69
Input Distribution Model 70
Print Capture 71
Choosing Visuals 72
Interaction with Other Extensions 73
Xlib Interface 73
XSolarisOvlPaintType 74
XSolarisOvlCreateWindow 74

XSolarisOvlIsOverlayWindow 75
XSolarisOvlSetPaintType 76
XSolarisOvlGetPaintType 77
XSolarisOvlSetWindowTransparent 78
XSolarisOvlCopyPaintType 79
XSolarisOvlCopyAreaAndPaintType 81
XReadScreen 86
Semantics of Existing Primitive Rendering Routines 88
Semantics of Existing Pixel Transfer Routine 89
`XGetImage` 89
XCopyArea and XCopyPlane 89
Portability Inquiry Routines 89
XSolarisOvlSelectPartner 89
XSolarisOvlSelectPair 96
Summary of New XLib Routines 99
8. Security Issues 101
Access Control Mechanisms 102
User-Based 102
Host-Based 102
Authorization Protocols 103
MIT-MAGIC-COOKIE-1 103
SUN-DES-1 103
Changing the Default Authorization Protocol 104
Manipulating Access to the Server 105

Client Authority File 106
Allowing Access When Using MIT-MAGIC-COOKIE-1 . . 107
Allowing Access When Using SUN-DES-1 107
Running Clients Remotely, or Locally as Another User 108
A. Multi-Buffering Application Program Interface, Version 3.2 109
Library File 109
Header File 110
New Routines 110
New Types 110
New Constants 110
New Structures 112
MBX Functions 113
Glossary 129
Index 135

Figures

Figure 1-1 OpenWindows Architecture 2

Figure 2-1 OpenWindows Server Architecture 12

Figure 4-1 The DPS Extension to X 29

Figure 6-1 Nonlinear Monitor Intensity Response 59

Figure 6-2 Gamma Correction 60

Tables

Table 1-1	OpenWindows Directories	7
Table 3-1	X11 Libraries	19
Table 4-1	DPS Libraries	31
Table 5-1	OpenWindows Font Formats	35
Table 5-2	Bitmap Font Formats	37
Table 5-3	Font Directory Structure	38
Table 5-4	Font File Availability	42
Table 6-1	Reference Display Devices Supported by OpenWindows	51
Table 6-2	Device Modifier Options	56
Table 7-1	XSolarisOvlCopyPaintType Source/Destination Combinations and Actions	80
Table 7-2	XSolarisOvlCopyAreaAndPaintType Possible Source/Destination Combinations and Actions	84

Preface

The *OpenWindows Server Programmer's Guide* provides detailed information on the OpenWindows™ server. It also provides an overview of the OpenWindows architecture and tells you where to look for more information.

This manual provides detailed information for software developers interested in interfacing with the OpenWindows server.

Who Should Use This Book

If you are interested in the components of the OpenWindows environment, read Chapter 1, "Overview of OpenWindows Architecture."

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 xvii). Read this manual if you need detailed information on the:

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

Before You Read This Book

Read *New Features* for important information about this release.

Check the following manuals for any corrections or updates to information in this manual:

- *SPARC: Installing Solaris Software*
- *x86: Installing Solaris Software*
- *Software Developer Kit Open Issues and Late-Breaking News*
- *Software Developer Kit Introduction*
- *Software Developer Kit Installation Guide*

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
- OPEN LOOK Intrinsics Toolkit (OLIT)
- 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, “Overview of OpenWindows Architecture” describes the architecture and the components of OpenWindows, including definitions, pointers to other documents, and the directory structure.

Chapter 2, “Introduction to the OpenWindows Server” describes the architecture of the OpenWindows server, the X and DPS extensions, Sun’s enhancements to MIT’s libraries and extensions, notes on color-related issues, and a list of applications you can run with the server.

Chapter 3, “X Features and Enhancements” contains an overview of the X Window System and discussions of its features and SunSoft’s enhancements to the sample server.

Chapter 4, “DPS Features and Enhancements” provides an introduction to the DPS system. It describes the DPS extension, how applications use DPS, SunSoft’s font enhancements to the DPS extension, DPS libraries, applications modified to use DPS, DPS security issues, and how to access information from Adobe, Inc.

Chapter 5, “Font Support” describes the core set of fonts provided and how to use and add fonts.

Chapter 6, “Visuals and Display Devices” describes the visuals and graphics devices supported. It provides hints for programming on advanced graphics devices and discusses multiple hardware colormaps and the installation of colormaps.

Chapter 7, “Visual Overlay Windows” describes visual overlay windows.

Chapter 8, “Security Issues” describes the security features of the OpenWindows environment.

Appendix A, “Multi-Buffering Application Program Interface, Version 3.2” describes the C language API (application program interface) to the MBX (Multi–Buffering) extension.

Related Books

For information on this release of OpenWindows, consult the following:

- *New Features*
- *Software Developer Kit Introduction*
- *Software Developer Kit Open Issues and Late-Breaking News*

For information on how to start up the OpenWindows environment, consult the following manuals:

- *SPARC: Installing Solaris Software*
- *x86: Installing Solaris Software*
- *Solaris Advanced User’s Guide*

To learn how to use the OpenWindows environment, consult the following manuals:

- *Solaris User’s Guide*
- *Solaris Advanced User’s Guide*

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

- *Desktop Integration Guide*
- *ToolTalk User's Guide*
- *OpenWindows 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

The following table 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 information tells you what is new in this release of the OpenWindows X server.

Visual Window Overlays

An overlay is a pixel buffer (either physical or software-simulated) into which graphics can be drawn. You can use overlays to display temporary imagery in a display window. See Chapter 7, “Visual Overlay Windows” for a description of the visual overlay API.

AccessX

This release of the OpenWindows server supports features compliant with the American Disabilities Act (ADA). This feature is available through an extension to the server, called AccessX.

Use the client program `accessx` to enable and disable the following capabilites provided by the AccessX extension: sticky keys, slow keys, toggle keys, mouse keys, bounce keys and repeat keys. Of these, the sticky, slow and mouse keys can be enabled using shift or other keys. The `accessx` client controls the toggle, bounce, and repeat keys and their settings.

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. This package is not installed automatically, unless you install the `All Cluster`.

pfb Fonts

The Openwindows server now supports Type 1 binary `.pfb` fonts.

Adobe NX Agent Support

The DPS client library `libdps` has been enhanced so that a client will automatically connect to a DPS NX agent running on your network, if it is unable to connect to the DPS extension. Your DPSX client need not be modified to take advantage of this support. The NX agent is available through Adobe.

Performance Enhancements

If you NFS mount your window server, mount it `setuid` allowable. This enables the X server to take advantage of performance features in the Solaris operating system.

Overview of OpenWindows Architecture 1

This chapter provides an overview of the OpenWindows window system architecture. The OPEN LOOK Graphical User Interface is briefly described, as well as the components of the OpenWindows product. This chapter also includes an overview of the file system.

Architecture Overview

Figure 1-1 on page 2 illustrates the architecture of the OpenWindows environment. Each rectangle bordered by a solid line is a component of the OpenWindows architecture. The “Operating System” and “Hardware” rectangles, bordered by dashed lines, are included in the figure to show you how the OpenWindows environment fits into a computer system. These two rectangles are *not* part of the OpenWindows architecture.

1

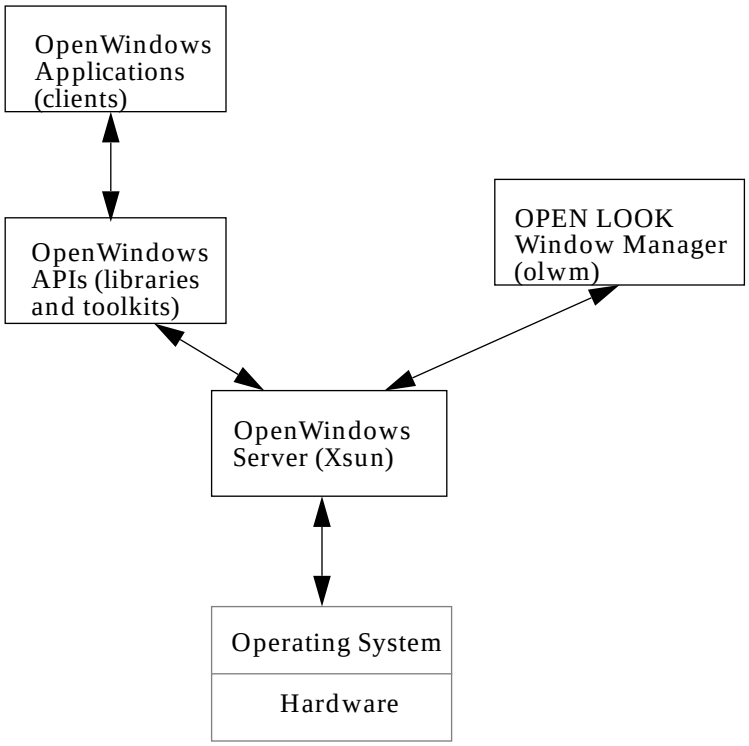


Figure 1-1 OpenWindows Architecture

A user interface specifies a consistent screen layout that effects applications (clients), APIs (Application Programmer's Interface), and window managers. The OPEN LOOK Graphical User Interface is the user interface implemented in the OpenWindows environment and is discussed on page 3.

This manual is primarily concerned with the OpenWindows server. Other architecture components are discussed in detail in separate technical manuals referenced throughout this chapter.

OPEN LOOK Graphical User Interface

The OPEN LOOK Graphical User Interface (GUI) is not software; it is a specification for a user interface that builds on the graphical user interface design pioneered by the Xerox Corporation. The OPEN LOOK GUI specifies windows and menus with common graphic symbols (instead of typed system commands) to provide an intuitive environment. Users are presented with a consistent screen layout that can be used across various platforms and operating systems. This common look and feel benefits end users, application programmers, and software vendors.

While the application programmer can implement the OPEN LOOK GUI from scratch by using Xlib, the usual implementation approach is to use a *toolkit* written for a specific windowing UI (for example, OLIT, or XView™). These toolkits provide routines that implement interface elements, allowing the programmer to concentrate on the design of the application. Examples of interface elements are windows, resize corners, and scrollbars.

For more information on the OPEN LOOK GUI, see the *OPEN LOOK Graphical User Interface Application Style Guidelines* available through SunExpress or your local bookstore.

OpenWindows Applications

OpenWindows applications (clients) are applications that run within the OpenWindows environment. These clients are implemented with the OPEN LOOK GUI and are integrated with other applications on the desktop. These applications are usually visible on the desktop as icons.

For more information on integrating your application with other OpenWindows application, see the *Desktop Integration Guide*.

For more information on how to use OpenWindows applications, see the *Solaris Advanced User's Guide* and the *Solaris User's Guide*.

The following applications are part of the OpenWindows environment:

- DeskSet Applications

  The DeskSet™ is a group of productivity applications, such as Mail Tool, File Manager, Calendar Manager, Clock, Print Tool, Audio Tool, and Icon Editor.

For more information on DeskSet, see the *Solaris User's Guide* and the *Solaris Advanced User's Guide*.

- Help Viewer

  Use the Help Viewer application to read help handbooks and a tutorial called *Introducing Your Desktop*. Help handbooks provide quick-reference information about the workspace and the DeskSet applications. The desktop tutorial provides an overview of using your workspace, including how to use the mouse, icons, windows, and menus.

  Choose Help from the Workspace menu to run the Help Viewer.

- Demo Applications

  Demonstration applications, or Demos that you can run are located in `/usr/openwin/demo`. These Demos are written with several OpenWindows toolkits. The source for some Demos is located in `/usr/openwin/share/src`.

Note – This directory is included by default only on systems installed with the "entire" configuration due to disk space limitations.

OpenWindows Application Programmer's Interfaces

An application programmer's interface (API) is a set of software routines you use to build applications. Toolkits and libraries are examples of APIs that range from high-level abstractions (toolkits and libraries such as XGL™ and SunPHIGS™) to low-level abstractions (libraries such as Xlib, the lowest level interface to the X Window System available to you).

OpenWindows Toolkits

The OpenWindows toolkits provide a set of user interface objects and widgets (such as scrollbars, menus, and buttons) that implement the look and feel of an OPEN LOOK application. The toolkits also provide a mechanism to manage the interpretation of events received from the window server. The application developer combines the interface objects and event handling mechanisms with application-specific code.

OpenWindows provides two toolkits and many applications built with these toolkits. Since these toolkits support the OPEN LOOK GUI, the OpenWindows applications and customer applications built with these toolkits are OPEN LOOK applications.

The two toolkits are:

- OPEN LOOK Intrinsics Toolkit (OLIT)
- XView

OLIT

The OPEN LOOK Intrinsics Toolkit (OLIT) is a GUI toolkit for OpenWindows that is based on the Xt Intrinsics from MIT (Massachusetts Institute of Technology). OLIT features an OPEN LOOK widget set (prebuilt user interface components).

For more information on programming with OLIT, see the following manuals:

- *OLIT QuickStart Programmer's Guide*
- *OLIT Reference Manual*
- *R5 Xt Toolkit Intrinsics Reference Manual*
- *The X Window System: Programming and Applications with Xt*

XView

The XView toolkit (X Window System-based Visual/Integrated Environment for Workstations) simplifies application development under the X Window System by providing you with a set of user interface objects. XView is based on several fundamental principles of object-oriented programming.

For more information on programming with XView, see the following manuals:

- *XView Programming Manual*
- *XView Reference Manual*
- *XView Developer's Notes*

Libraries

Libraries are files that incorporate collections of software routines. Link these files into your programs as needed. Libraries provide software reusability.

1

All of the libraries available in OpenWindows are in `/usr/openwin/lib`. See “OpenWindows Directory Structure” on page 7 for a list of libraries available and “X11 Libraries” on page 19 and “DPS Libraries” on page 31 for further information on supported libraries.

OPEN LOOK Window Manager

A window manager implements the functions with which a user can control the appearance of windows on the screen. These functions include moving, resizing, opening, closing, and quitting windows. Other functions of window managers include setting input focus, installing colormaps, and starting up new applications.

By default, OpenWindows runs the OPEN LOOK window manager (`olwm`). It is designed to manipulate windows using a two- or three-button mouse.

X Window System managers that are ICCCM (Inter-Client Communication Conventions Manual) compliant (for example, `gwm`, `mwm`, `twm`) are compatible with OpenWindows, and `olwm` is compatible in a generic X11 environment. For information on how to change from the default `olwm` to another X window manager, see the `olwm`(1) man page.

The X Consortium’s ICCCM specifies how a client coexists properly with other clients sharing the same server. For more information on the ICCCM, see “ICCCM Compliance” on page 21 and the *X Protocol Reference Manual*.

For more information on window managers, see the following documentation:

- *Xlib Programming Manual*
- `olwm`(1) man page

OpenWindows Server

The OpenWindows server (called `XSun`) is a program that is the foundation for the OpenWindows environment. It is the X Window System server (Version 11, Release 5–X11R5) with a Display PostScript™ (DPS) imaging system extension. It implements a client-server model of window systems, is portable to a wide variety of hardware platforms, and supports portable X11 extensions.

The OpenWindows server is described in more detail in the remaining chapters of this manual.

OpenWindows Directory Structure

A software product's directory structure can show a great deal about the product's features. It can tell you where the executables, include files, and libraries reside, and reveal the basic logic or design of the product. Table 1-1 helps you to get familiar with the OpenWindows product.

The OpenWindows directory structure includes the following top-level directories (all of which are prefixed by `/usr/openwin`).

Table 1-1 OpenWindows Directories

Directory	Subdirectory	Content
`SUNWits`		Server private files for Internal Use Only
`/bin`		OpenWindows executables
`/demo`		OpenWindows demonstration programs
`/etc`		Symbolic link to `/share/etc`
	`/keytables`	US and international keytables, and `keytable.map`
	`/tt`	ToolTalk data files
	`/workspace`	`/patterns` (`.xbm` files and attributes)
`/include`		Symbolic link to `/share/include` Various library header files
	`/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
	`/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

Table 1-1 OpenWindows Directories (Continued)

Directory	Subdirectory	Content
/lib		OpenWindows default start-up files and libraries, MIT core distribution libraries, rgb files
	/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		OpenWindows man pages
		Symbolic link to /share/man
	/man1	OpenWindows command man pages
	/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		Architecture-independent files
	/etc	Location of files in /etc
	/images	/PostScript, /fish, /raster
	/include	Location of files in /include
	/locale	Location of files in /lib/locale

Table 1-1 OpenWindows Directories (Continued)

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

1

Introduction to the OpenWindows Server 2

The OpenWindows server is a program that is the foundation for the OpenWindows environment. It is MIT's X11R5 sample server with a Display PostScript (DPS) imaging system extension and considerable Sun added value. The OpenWindows server also includes several X extensions and DPS font enhancements.

Throughout this document, *server* is used interchangeably with OpenWindows server, and *sample server* is used interchangeably with MIT's X11R5 sample server.

The following topics are discussed in this chapter:

- Server architecture
- X11R5 server, its layers and the font management library
- MIT's X extensions
- DPS extension
- Types of applications you can run with the server

For more information on the server, see the books listed in "Related Books" on page xvii and the following manual pages:

- `Xsun`(1) OpenWindows server
- `Xserver`(1) MIT's sample server
- `openwin`(1) OpenWindows start up command

2

Server Architecture

Figure 2-1 illustrates the structure of the server. This diagram is an expansion of the OpenWindows Server architecture component in Figure 1-1 on page 2.

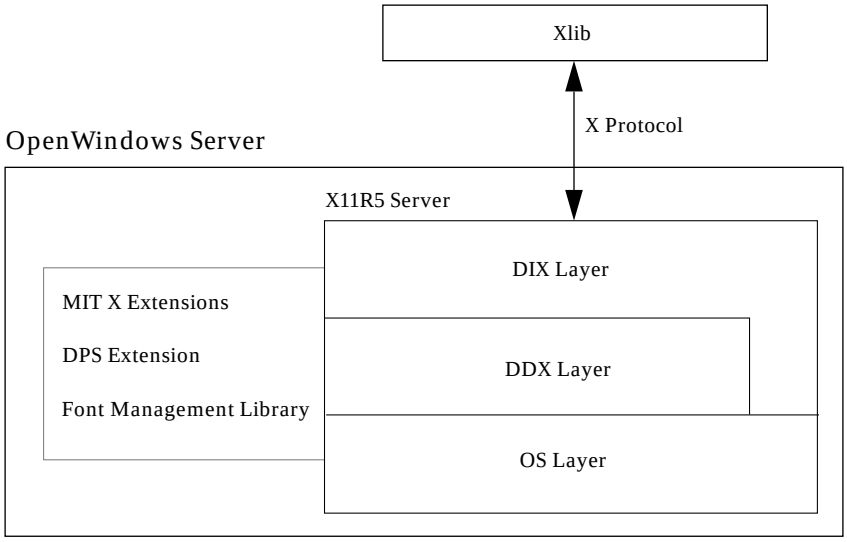


Figure 2-1 OpenWindows Server Architecture

X11R5 Server

The X11R5 sample server from MIT was designed and implemented to be *portable*; it hides differences in the underlying hardware from applications, or clients. The 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)

  The DIX layer contains functions that do not depend on graphics hardware, input devices, or the host operating system—these functions are portable from one implementation to another. It dispatches client requests, manages the event queue, distributes events to clients, and manages visible data structures.

- Device-Dependent Layer (DDX)

  The DDX layer contains routines that depend on graphics hardware and input devices the server must accommodate. This layer is a major portion of the OpenWindows server, Sun's value-added product. This layer includes creating and manipulating 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.

- Operating System Layer (OS)

  The OS layer contains functions that depend on the host operating system. This layer is another part of Sun's value-added product. The OS layer manages client connections and connection authorization schemes, and provides routines for memory allocation and deallocation.

- 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 font features of the OpenWindows server are described in detail in Chapter 5, "Font Support."

MIT X Extensions

The X Window System is extensible; that is, new features and technology can be added easily. The OpenWindows server supports six X extensions as defined (or proposed) by the MIT X Consortium. For more information on the standard X Extension Mechanism, see *The X Window System Server* and the *Xlib Programming Manual.*

The OpenWindows server supports the following MIT extensions:

- XInput
- MBX (Multi-Buffering)
- SHAPE
- MIT-SHM (Shared Memory)

- XTEST
- MIT-SUNDRY-STANDARD

See "MIT X Extensions Supported" on page 21 for more information on these extensions.

DPS Extension

The DPS extension is implemented as an extension to the server. Clients send PostScript to the server via *wraps*. Data can be returned from the server to the client by specifying *output* arguments. The DPS client library implements DPS client-server communication transparently using the X protocol.

Applications that use the DPS extension create one or more *contexts*. A context can be thought of as a virtual printer, with its own stacks, input/output facilities, and memory space.

The interpreter schedules context execution. Each context has access to its private *VM* (virtual memory space). An additional common portion of VM space, called *shared VM*, is shared among all contexts and contains system fonts and other shared resources. *Private VM* can contain fonts private to the context.

See Chapter 4, "DPS Features and Enhancements" for more information on this extension.

Applications That Run With the Server

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

- Applications written with the following toolkits:
  - OpenWindows 3.3 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 *Solaris 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

2

X Features and Enhancements 3

This chapter discusses the following X-related topics:

- Overview of the X Window System
- X11 Features Supported
- X11 Features Not Supported
- ICCCM Compliance
- MIT X11 Extensions Supported
- Notes on X11 Programming

OpenWindows is based on and complies with the features present in MIT's X11 Release 5 (X11R5). In addition, Sun has added several enhancements that let programmers create more sophisticated applications.

Overview of the X Window System

The *X Window System*, or *X*, is a client-server, network-based windowing system developed at MIT (Massachusetts Institute of Technology) in 1984. Several versions of X have been developed. (X and X11 are used interchangeably throughout this chapter.)

X has been adopted as an industry-standard window system and is supported by the MIT X Consortium, which was formed in 1988. This consortium currently consists of over 65 companies, several universities, research organizations, and international vendors. The X Consortium is self-supporting

and was formed to promote cooperation within the computer industry in creating standard software interfaces at all layers in the X Window System environment.

The X Window System's architecture is based on the client-server model. The X server runs on a computer with a screen, keyboard, and pointing device (usually a mouse). Applications that run in the X environment are *clients* of the server. Clients can coexist on the computer with the server or they can be located on a remote system somewhere on the network.

Individual windows are controlled by a special client program, the window manager. The OpenWindows environment includes the `olwm` window manager. See "OPEN LOOK Window Manager" on page 6 and the `olwm`(1) man page for more information on the capabilities of `olwm`.

There are numerous books on all aspects of X and the X Window System. See the preface for a list of recommended books available through SunExpress and your local book store.

X Protocol

Clients and the server exchange information using a well-defined communication protocol—the *X protocol*—that forms the basis of the X Window System. Clients use the X protocol to send requests to the server to create and manipulate windows, to generate text and graphics, to receive input from the user, and to communicate with other client applications. The server uses the X protocol to send information back to clients in response to their requests and to deliver keyboard and other user input to the clients in the form of specialized data structures called *events*.

The X Library

The X protocol is implemented with a standard library of routines called Xlib, the X library. It provides you with a procedural interface that hides many of the low-level details of the X protocol, such as message formats. Various functions are also included that are not protocol-related, but important when building applications. The exact interface for this library may differ for each programming language. The OpenWindows development environment provides Xlib with a C programming language interface.

X Toolkits

X toolkits are one level up in terms of ease-of-use from the Xlib routines, just as the Xlib routines are one level up from the X protocol. Toolkits and higher-level graphics libraries can be implemented on top of Xlib and usually call Xlib directly.

The X protocol does not specify the look and feel of applications written for X nor how they should respond to user input. Although Xlib provides the foundation, write most of your applications using toolkits and graphics libraries that provide a consistent look and feel characteristic of a particular windowing system.

The X toolkits bundled with the OpenWindows product are XView and OLIT (OPEN LOOK Intrinsics Toolkit). See “OpenWindows Toolkits” on page 4 for more information.

X11 Features

The following libraries and applications, most of which are available from the MIT X Consortium, run on the OpenWindows server and are supported by Sun.

X11 Libraries

The X libraries are listed in the following table. The `.so` and `.a` files that comprise these libraries are located in `/usr/openwin/lib`.

Table 3-1 X11 Libraries

Library	Description	Available From MIT	Sun’s 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

Table 3-1 X11 Libraries (Continued)

Library	Description	Available From MIT	Sun's Value Added
`libXinput`	Binary compatibility library for previous input extension	No	Sun library
`libXi`	Xinput Extension library	Yes	Bug fixes Supports OpenWindows X extensions
`libXmu`	X Miscellaneous Utilities library	Yes	Search path includes `/usr/openwin`
`libXol`	OLIT library	No (Available from USL)	Sun product—see the preface for a list of OLIT manuals
`libXt`	Xt Intrinsics library	Yes	Includes all private X Consortium patches as of 12/9/92
`libxview`	XView library	Yes	Sun product donated to X Consortium Bug fixes not included in MIT's X11R5 `libxview`

Supported X11 Applications

The OpenWindows environment includes the following client applications available from the MIT X Consortium:

- `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 MIT X Consortium, that run on the OpenWindows server but are *not* distributed or supported by Sun:

- Andrew, InterViews
- The `uwm` and `wm` window managers
- The CLX Common Lisp interface
- "contrib" MIT clients

ICCCM Compliance

OpenWindows is fully compliant in all areas of the ICCCM (Inter-Client Communication Conventions Manual). For more information on the ICCCM, see the *X Protocol Reference Manual.*

MIT X Extensions Supported

The OpenWindows server supports six X extensions as defined, or proposed by the MIT X Consortium. The server also includes OpenWindows-specific X extensions; however, they are not intended for use by client programs.

For more information on the standard X Extension mechanism, see *The X Window System Server* the *Xlib Programming Manual.*

The OpenWindows server supports the following MIT extensions:

- XInput
- MBX (Multi-Buffering)
- SHAPE
- MIT-SHM (Shared Memory)
- XTEST
- MIT-SUNDRY-NONSTANDARD

3

How To Access MIT X11 Extension Standards

The MIT X Consortium X11 standards referenced in the following sections are readily available to systems on Internet. The MIT X11 documentation resides in the `/pub/R5untarred/mit/doc/extensions` directory on the `ftp.x.org` machine. Use the File Transfer Program (`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.

In the following sections the specification name for each extension is given, as well as the associated file name (on `ftp.x.org`) in parenthesis.

XInput Extension

The XInput Extension is Sun's implementation of the MIT 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 from each other and independently from the core devices.

MBX (Multi-Buffering) Extension

The Multi-Buffering Extension is Sun's implementation of the MIT 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 A, "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. Backwards compatibility is not guaranteed.

SHAPE Extension

The SHAPE Extension is Sun's full implementation of the MIT 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.

MIT-SHM (Shared Memory) Extension

The Shared Memory extension is Sun's full implementation of the MIT X Consortium experimental *The MIT 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 MIT 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.

MIT-SUNDRY-NONSTANDARD

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 the bug-compatibility mode on (handles certain erroneous requests) or off (returns an error for erroneous requests) and a request that gets the current state of the mode. This extension can be dynamically turned on or off with `xset`, or at server start up with `openwin`. See the `xset`(1) and `openwin`(1) man pages, specifically the `-bc` option, for more information.

3

Notes on X11 Programming

Common X11 programming issues are discussed below.

Compose Key Support

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

---

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

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 the programming task. 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 MIT. `rgb.txt` is compiled into the `dbm`(3) database files, `rgb.dir` and `rgb.pag`. When the server first 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 programmers who intend to use the OpenWindows 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 – It is important that programmers not rely on Black and White being in certain pixel locations. Future versions of the OpenWindows server and the servers of other vendors may have these colors located in different positions than the current server. For maximum portability and compatibility, X11 clients should always be written to 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 the “Hints for Windows Programming With Visuals” on page 57.

3

DPS Features and Enhancements 4

This chapter briefly discusses the following Display PostScript (DPS) system topics:

- Introduction to the DPS system
- How applications use DPS
- DPS extension
- OpenWindows' font enhancements to DPS
- DPS Libraries
- Applications modified to use DPS
- DPS security issues
- Accessing information From Adobe
- When DPS encounters internal errors

See Adobe's *Programming the Display PostScript System with X* for more detailed information.

Introduction to the DPS System

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]

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.

4

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.

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 supported display without modification to the application program.

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. 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.

How Applications Use the DPS System

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. The PostScript interpreter switches among contexts, giving multiple applications access to the interpreter.
2. The application then 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.

DPS Extension to X

The X Window System is extensible; that is, new features and technology can be added easily.[1] The Display PostScript system is implemented as an extension to the X Window System; the extension is sometimes referred to as *DPS/X*. Figure 4-1 shows the components of DPS and their relationship to X.

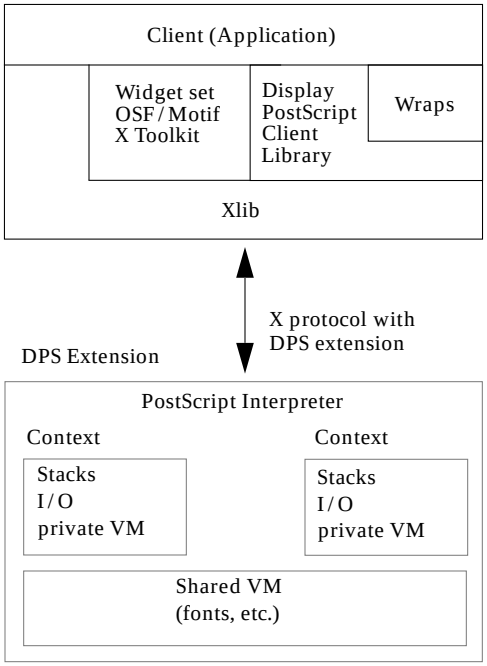


Figure 4-1 The DPS Extension to X

The DPS extension is implemented as part of the X Window System client-server network architecture. 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 or *wraps*. Data can be returned from the server in the form of *output* arguments. The Client Library implements DPS client-server communication transparently using the low-level communication protocols provided by the X Window System.

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.

4

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.

The interpreter handles the scheduling associated with executing contexts in time slices. Each context has access to a private portion of PostScript *VM* (virtual memory space). 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.

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. *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.

DPS Font Enhancements

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

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

See Chapter 5, “Font Support” for more information.

DPS Libraries

The DPS libraries are listed in the following table. The `.so` and `.a` files that comprise these libraries are located in the `/usr/openwin/lib` and `/usr/openwin/lib/libp` directories.

Table 4-1 DPS Libraries

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

For information on these libraries, see *Programming the Display PostScript System with X* and *PostScript Language Reference Manual*.

Adobe added Adobe NX agent support to `libdps` for OpenWindows 3.4; it is described in the following section.

Adobe NX Agent Support

The Context creation routines (XDPSCreateSimpleContext and XDPSCreateContext) in `libdps` now 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 window startup process.

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

Applications Modified to use DPS

The PageView, ImageTool, and DocViewer applications have been modified to use DPS. If your application calls one of these applications, it will run in the current OpenWindows environment.

4

DPS Security Issues

The OpenWindows environment provides, and in some cases exceeds, MIT's X11R5 sample server security levels. To ensure this, DPS programmers should be aware of two DPS-specific security features: PostScript file operators' inability to access system files, and secure context creation.

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 OpenWindows environment, you cannot—by default—use PostScript file operators to open or otherwise access a system file.

For applications, the client should perform necessary file operations, rather than the server. This prevents a client with a user id different than the server's from accessing the server's file access privileges. 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.

Secure Context Creation

DPS contexts normally have access to global data that provide a mechanism for inserting "Trojan horses." This allows a context to peer 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 OpenWindows 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`:

```
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 will generate an Access error.

How to Access Information From Adobe

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

If you have access to Internet, use the File Transfer Program (`ftp`) to download files from the `ftp.mv.us.adobe.com` machine. Read the `README.first` file for information on the archived files. See the “Public Access File Server” section in the preface of *Programming the Display PostScript System with X* for details on how to obtain this information by electronic mail.

4

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 OpenWindows 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)
```

Font Support 5

The OpenWindows server provides robust 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 OpenWindows environment provides 55 Latin fonts for European text and two symbol fonts. You can add other fonts to the system if you want.

Font Formats

Fonts from different vendors come in different formats. Table 5-1 lists the various font formats, their vendors, and the associated file types supported by the OpenWindows environment.

Table 5-1 OpenWindows Font Formats

Font Format	Vendor	File Types
F3 (Type 7)	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`
Portable Compiled Format	MIT	`.pcf`
Bitmap Distribution Format	Adobe	`.bdf`

Table 5-1 OpenWindows Font Formats (Continued)

Font Format	Vendor	File Types
Big Endian Prebuilt Format	Adobe	`.bepf`
Little Endian Prebuilt Format	Adobe (for x86 only)	`.lepf`
Server Natural Format	MIT	`.snf`
Old OpenWindows Bitmap	SunSoft	`.fb`

The fonts provided by the OpenWindows server are located in the `/usr/openwin/lib/X11/fonts` directory. For more information on the directory structure see "Font Directory Structure" on page 38.

Outline and Bitmap Fonts

OpenWindows supports two types of font representation: *outline* fonts and *bitmap* fonts. In the X11 server, outline fonts can be scaled to any desired size; in Display PostScript (DPS) 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 very frequently, they are also kept in separate files in pre-rendered bitmap form. This saves the server from having to scale and render them. However, the resulting bitmap fonts can only be displayed in 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 between outline and bitmap fonts, 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 font from `.bdf` format to Portable Compiled Format (`.pcf`)

- `bdftosnf` Converts `.bdf` files to Server Natural Format (`.snf`) files

As illustrated in Table 5-2, 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 5-2 Bitmap Font Formats

Font Format	File Extension	Binary	Architecture-specific
Bitmap Distribution	`.bdf`	No	No
Portable Compiled	`.pcf`	Yes	No
Server Natural Format	`.snf`	Yes	Yes
Old OpenWindows Bitmap	`.fb`	Yes	Yes
Little Endian Prebuilt Format	`.lepf`	Yes	Yes (x86)
Big Endian Prebuilt Format	`.bepf`	Yes	Yes (SPARC)

The OpenWindows 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.

Replacing Outline Fonts with Bitmap Fonts

The OpenWindows environment automatically replaces some outline fonts by 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.

When Replacement Occurs

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.

5

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 40 for details.)

Locating Fonts

By default, the OpenWindows server looks for fonts in directories under the `/usr/openwin/lib/X11/fonts` directory. See Table 5-3 for the complete font directory structure.

Font Directory Structure

The directories below are preceded by `/usr/openwin/lib/X11/fonts`.

Table 5-3 Font Directory Structure

Directory	Subdirectory	File Suffixes	Contents
`/100dpi`		`.pcf`	Bitmap fonts
`/75dpi`		`.pcf`	Bitmap fonts
`/F3`		`.f3b`	F3 format outline fonts
	`/map`	`.map`	F3 character set specifications
`/F3bitmaps`		`.pcf`	Bitmap fonts
`/Speedo`		`.spd`	Bitstream Speedo format outline fonts

Table 5-3 Font Directory Structure (Continued)

Directory	Subdirectory	File Suffixes	Contents
/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 this default either at start up or after the server has been started.

At start up, use:

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

where 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.

After the server has started, you can use either the `xset` command or `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
```

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 `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.

The `xset` command:

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

prepends */dir1/dir2/fonts* to the X11 font path. (Use `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 command:

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

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

The following `openwin` command:

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

appends /dir1/dir2/fonts to the DPS resource path.

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 – A server reset clears both the X11 font path and the DPS resource path.

Font File Suffixes

The OpenWindows environment is configured so that most X11 fonts are also available in DPS (see Table 5-4 below). DPS supports a slightly different set of fonts than those supported by X11.

Table 5-4 Font File Availability

Font Description	Font File Suffix	Available in X11	Available in DPS
Type1 outline fonts	`.pfa` (scaled)	Yes	Yes
Type1 outline fonts	.pfb (scaled)	No	Yes
Big Endian Prebuilt Format	`.bepf`	No	Yes
Little Endian Prebuilt Format	`.lepf`	No	Yes
F3 (Type 7)	`.f3b` (scaled)	Yes	Yes
Old OpenWindows Bitmap	`.fb`	Yes	No
Speedo	`.spd` (scaled)	Yes	No
Type 3	`.ps` (scaled)	No	Yes
Portable compiled	`.pcf`	Yes	Yes
Bitmap Distribution	`.bdf`	Yes	No
Server Natural Format	`.snf`	Yes	No

Associated Files

The OpenWindows 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

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 eight-bit fonts. Multibyte 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 44.

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 to map the long internal XLFD font names to shorter names that are easier to enter on a command line.

   Here is a sample `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.

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 check to see if the server recognizes your new fonts.**
   `xlsfonts` lists all the names of all fonts that are accessible to the window server.

Adding Outline Fonts

Follow the steps included in this section to install outline fonts. The OpenWindows 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 run** `makepsres` **in the** */newfonts* **directory.**
   This creates a `PSres.upr` file. The system requires this file if you want to use these fonts within the DPS extension.

2. **If you are installing F3 fonts, create a** `.upr` **file.**
   Use the template below for the .upr file. Replace the example values given below 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`**) or Speedo (**`spd`**) fonts, create a** `fonts.scale` **file.**
   The `fonts.scale` file contains the mapping of an internal X11 font name to an easily understood font name. The fonts.scale file will be 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
```

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 additional information on the XLFD font naming convention.

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

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` at this point.

```
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 which 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 check if the server recognizes your new fonts.** `xlsfonts` lists all the names of all fonts that are accessible to the window server.

You can now use the fonts in the */newfonts* directory in your applications. In X11, you do not need to restart the OpenWindows server since `xset` dynamically changes the font path. See “Changing the Default Font Path in X11” on page 39 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.

5

Visuals and Display Devices 6

This chapter defines some of the terms used to describe how client applications and the OpenWindows server interact to control what is displayed on a user's screen. Multiple hardware colormaps and advanced display devices are discussed, and some important programming hints are provided.

The X11 server requires you to take responsibility for ensuring your applications run properly on a wide variety of machine configurations.

Display Devices

The computer monitor is connected to a *display device* (also called a *graphics adapter*) that controls what is shown on the screen. The display device has memory dedicated to storing display information.

Reference Display Devices

The OpenWindows server treats certain display devices as *reference devices.* This means that example device handlers for these devices are provided in the Solaris DDK (Driver Developer Kit). These devices are described in more detail in "Reference Devices and Visuals" on page 51.

SPARC *Supported Reference Devices*

The SPARC reference devices supported by the OpenWindows server are:

- BW2
- CG3
- CG6
- CG8

x86 *Supported Reference Devices*

The x86 reference devices supported by the OpenWindows server are:

- vga4
- vga8
- 8514

IHV Display Devices

In addition to the reference devices, the OpenWindows server supports any device for which a valid device handler is written and configured into the system. The process of writing and configuring a device handler is described in the *OpenWindows Server Device Developer's Guide,* which is included in the Solaris DDK product.

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.

Multiple Depth Devices

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 most MPG devices, windows can be created using any of the exported visuals.

Default Visual

For each X11 screen, there is one special visual that is designated the *default visual*. This is the visual assigned to the screen's root window and is, unless otherwise specified by the client, the visual which client window's are assigned. See "The Default Visual" on page 55 for more information.

Reference Devices and Visuals

This section describes in greater detail the reference display devices supported and the visuals they export.

Reference Devices

The reference display devices supported by OpenWindows are listed in Table 6-1.

Note – Throughout this chapter "n/a" means not applicable.

Table 6-1 Reference Display Devices Supported by OpenWindows

Product Name	Device Name	Device Driver	Bus	Exported Depths
n/a	BW2	`/dev/fbs/bwtwo`*X*	SBus, VME/obio, P4	1-bit
n/a	CG3	`/dev/fbs/cgthree`*X*	SBus	8-bit
GX	CG6	`/dev/fbs/cgsix`*X*	SBus, P4	8-bit
GXplus/ TurboGXplus	CG6	`/dev/fbs/cgsix`*X*	SBus	8-bit
TC	CG8	`/dev/fbs/cgeight`*X*	SBus, P4	1, 24-bit (MPG)
VGA	vga4	`N/A`	ISA, EISA, MCA	8-bit
VGA	vga8	`N/A`	ISA, EISA, MCA	8-bit
8514/A	i8514	`N/A`	ISA, EISA, MCAS	8-bit

Product Name

The product name is commonly used to identify the type of display card.

Device Name

The device name is used to specify the display adapter to OpenWindows.

Note – 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).

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 has two CG3s, the first would be `/dev/fbs/cgthree0`, and the second would be `/dev/fbs/cgthree1`. If a system has one CG3 and one GX, the CG3 would be `/dev/fbs/cgthree0` and the GX `/dev/fbs/cgsix0`. The server is configured to support a maximum of 16 displays; the limitations you might encounter are the number of framebuffers your hardware supports.

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.

Exported Depths

These are the depths of the visuals advertised by the server for screens of this particular device type. MPG (Multiple Plane Groups) indicates that this device supports multiple depth visuals.

SPARC *Device-Specific Information*

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 visual is the built-in default visual. A variety of BW2 frame buffers is available for different buses and different 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 below). The built-in default visual is 8-bit PseudoColor.

GX and GXplus (CG6)

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 visual graphics APIs. Several 8-bit visuals are supported (see below) 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 OpenWindows server uses the GXplus to automatically accelerate X11 pixmaps by using offscreen storage whenever possible.

Note – This chapter treats the GXplus as a GX.

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 Default Visual” on page 55 describes how to change the default if a color default visual is desired.

The plane groups of the TC do not conflict with each other; they are completely separate memory buffers. OpenWindows, by default, 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*. This behavior may be disabled by using the `-nominexp` option of `openwin`(1). If this option is used, 1-bit windows will damage 24-bit windows and 24-bit windows may damage 1-bit windows.

The OpenWindows server also provides minimized exposure for other IHV 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 *Device-Specific Information*

VGA

The VGA is a simple color dumb frame buffer. The server supports VGAs 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. 4-bit mode is often limited to a resolution of 640x480 as this is the basic VGA graphics mode that is available on all VGA devices. Most VGAs provide a bitsPerRGB of 6.

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 will automatically move the physical window in the virtual display in the direction that the mouse was pushed until the physical window hits the edge of the virtual boundary.

Use panning only if you are an experienced OpenWindows user. Icons, pop-up boxes (menus / dialogs etc.) 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.

The vga8 server is also capable of supporting the XGA as a dumb frame buffer.

8514/A

The 8514/A is an 8-bit indexed color graphics accelerator providing general window system acceleration. It provides substantially improved performance compared to a VGA. The server limits its support of 8514/As 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/As provide a bitsPerRGB of 6.

The Default Visual

At all times, a default visual exists. The *default visual* of an X11 screen is one of the exported visuals for the screen. When a client application is executed, its windows are assigned the default visual unless it specifies non-default or *alternate* visuals.

The *built-in default visual* is the visual hard-coded in the OpenWindows server. This is the default visual unless you specify a different default visual when you run `openwin`(1). The built-in default visual for each screen varies with the characteristics of the screen's display device.

The *screen default visual* is the visual advertised to clients in the connection block. This is the built-in default visual unless you specify a different supported visual to be the default when you start up OpenWindows.

An *allowable default visual* is a supported visual that can be the screen default visual.

Changing the Screen Default Visual

At times, it may be desirable to change the default visuals that the X11 window server advertises in the X11 connection block. One possible reason is to force client programs that cannot run in alternate or non-default visuals to run in a selected visual. You should be careful when using this mode because the default visual can have a subtle effect on the behavior of many client programs. Often, client programs are unable to deal with visuals of some depths or classes, especially those with 24-bit depths.

The default visual and the list of supported visuals exported by the server can be examined from X11 using `XGetVisualInfo`(3).

To make an allowable default visual the screen default visual, the class and depth of the visual must be specified on the `openwin` command line with the `-dev` option:

```
example% openwin -dev <device modifier options>
```

Table 6-2 describes the available device modifier options pertaining to the screen default visual.

Table 6-2 Device Modifier Options

Device Modifier Options	Description
`defclass` <*class*>	This option uses the specified visual as the default visual. The default is device-dependent. The legal values are: GrayScale, StaticGray, PseudoColor, StaticColor, DirectColor, and TrueColor.
`defdepth` <*n*>	This option selects the depth of the screen default visual. *n* is an integer that specifies the depth. The default is device-dependent.
`grayvis`	This option indicates the screen default visual is to have a gray class (StaticGray or GrayScale).

The `defclass` option is used to determine the visual class of the screen default visual, and may be used by itself or in combination with `grayvis`.

If `grayvis` is specified, color visuals are not supported by the server. After using the `grayvis` option, StaticColor and PseudoColor are not returned by `XGetVisualInfo`(3).

SPARC *Example*

As an example, the following command line runs OpenWindows on the GX device with an 8-bit StaticGray screen default visual.

```
example% openwin -dev /dev/fbs/cgsix0 grayvis
```

x86 *Example*

As an example, the following command line runs OpenWindows on the VGA device with an 8-bit GrayScale screen default visual.

```
example% openwin -dev vga8 grayvis
```

Troubleshooting/Error Messages

- If the device does not support the requested visual, the following error message is returned. (# represents the depth number requested and *n* represents the requested display device.)

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

  If this message is returned for a supported visual/device combination as indicated in Table 6-1 on page 51, then an installation problem exists.

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

Hints for Windows Programming With Visuals

This section discusses various issues that arise when programming X11 applications targeted to devices that support different visuals. In particular, programming for portability is discussed.

Default Visual Assumptions

A common mistake in programming an X11 client is assuming 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.

Here are some common mistakes:

- 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`

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

Multiple Hardware Colormaps

The OpenWindows environment also supports devices with multiple hardware color lookup tables (LUTs). Multiple color LUTs are provided on some devices to reduce *colormap flashing*. This is a visual effect (sometimes called *technicolor*) which happens when the pixels of one window are displayed with the colors of another window, because there are not enough simultaneously displayable colors on the device. Multiple color LUTs increases the number of colors simultaneously displayable and, thus, alleviates colormap flashing.

This section describes information you should know about multiple hardware colormaps from an application programming and end user perspective.

Colormap Installation–Multiple LUT Devices

Ultimately, it is the window manager that decides which colormaps are installed in which LUTs. This is referred to as the *window manager colormap installation policy*. This policy is different from the server colormap installation policy. The server colormap installation policy only specifies how the server reacts to various `InstallColormap` and `UninstallColormap` requests; the window manager colormap installation policy defines how and when a window manager will send these requests to the server.

Colormap Demo

The program `/usr/openwin/bin/xcolor` provides a convenient way of visually understanding what is happening to your hardware LUT. It always displays the most recently installed colors in the hardware colormap, arranged in a 2D array. Each row has 16 colors. Pixel 0 is in the upper left-hand corner and pixel 255 is in the lower right-hand corner. This demo program is very useful for understanding colormap flashing behavior.

Note – Do not confuse what you see with the contents of the default software colormap. The default software colormap will be seen only if it is installed. Additionally, extra colors may be present in the hardware colormap that are not allocated in the default colormap because of installation rules.

Note – `xcolor` is only useful for viewing the LUT on a single LUT device. On a multiple LUT device, the LUT viewed may change in unpredictable ways.

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. The actual light intensity response of CRT monitors follows a power function:

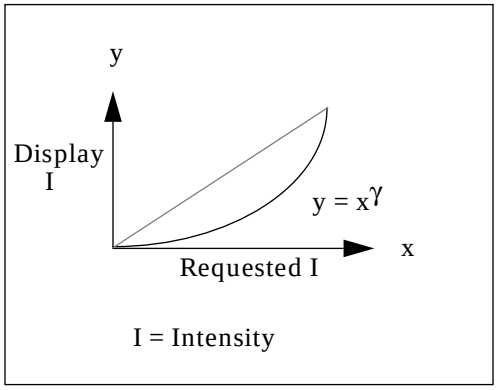


Figure 6-1 Nonlinear Monitor Intensity Response

The exponent of this function (γ) is called the visual's *gamma*. The dotted line in the figure represents the desired response. The solid line represents the actual response of the monitor. On Sun systems, gamma is usually 2.22 but, in general, it can vary slightly between monitors. Most liquid crystal display (LCD) monitors have a gamma of exactly 1.0.

Some devices support visuals that compensate for this darkening effect. This is called *gamma correction*. The ZX, a Sun 3D accelerator, is an example of a device that does this. This correction is done by altering colors coming out of the frame buffer with the inverse of the monitor's response.

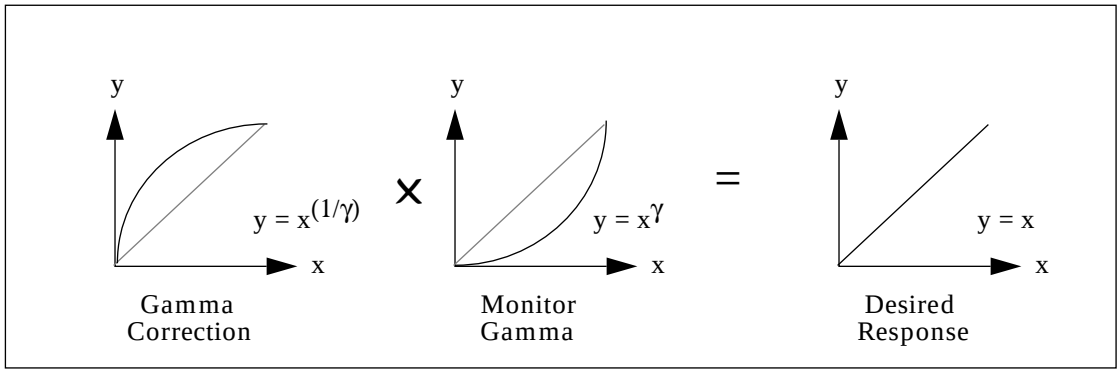


Figure 6-2 Gamma Correction

Refer to *Fundamentals of Computer Graphics*, Foley and Van Dam for a fuller discussion of gamma correction.

Because the overall intensity response 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.

Some applications require a linear visual to avoid visible artifacts. For example, an XGL 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 of this variety.

The linearity of default visuals on most devices is nonlinear. Therefore, linear applications should not depend on the default and should always explicitly search for a linear visual. See "Finding a Linear Visual" on page 61 for an example.

Note – In similar fashion, it is a good idea for nonlinear applications to explicitly search for a nonlinear visual. However, since this is typically the default on most devices, it is not as critical. But it is still a good policy to do so.

Finding a Linear Visual

Linearity is not a standard X11 attribute. However, it can be determined on Solaris by querying the visual's gamma. This is done by calling `XSolarisGetVisualGamma`(3). To use this routine, the application must be linked with the Solaris `libXmu`. 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%).

Code Example 6-1 on page 62 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.

Code Example 6-1 XGL 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);
}

 ...
}
```

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.

Visual Selection Alternatives

The above 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 GX, do not have any linear visuals. Other devices, for example ZX, have only a single linear 24-bit TrueColor visual. Other devices like SX (a Sun imaging accelerator) can support such a visual but don’t by default; they must be

reconfigured. Some other 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.

Support Level

`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.

Visual Overlay Windows 7

Certain applications can benefit from the ability to display temporary imagery in a display window. The users of these applications may wish to 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.

The use of *overlays* is a common technique that applications can use to achieve these effects. An overlay is a pixel buffer (either physical or software-simulated) into which graphics can be drawn. When the overlay is physical (i.e. 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.

This chapter presents a model for an application programming interface (API) that provides overlay capabilities in the Solaris Visual environment.

Basic Features of Overlay Windows

The following sections introduce the basic characteristics of overlay windows.

Definition

An overlay window is a special class of an X `InputOutput` window. 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`.

Standard X InputOutput windows can be rendered to with pixels of only one type of paint: opaque. Pixels painted opaquely obscure pixels in underlying windows. Opaque pixels have associated color values which are displayed.

The unique feature of overlay windows is that they permit pixels to be rendered with a new type of paint: *transparent paint*. Pixels rendered transparently have no intrinsic color; they derive their displayed color from whatever pixels lie beneath.

Both opaque and transparent paint can be rendered to an overlay window. Standard X windows and other drawables (such as pixmaps) only accept opaque paint.

Overlay windows are created using a new `XSolarisOvlCreateWindow` routine. Details are provided below.

Overlay windows are destroyed with `XDestroyWindow` or `XDestroySubwindows`.

Creating an Overlay Window

There is a new routine defined to create an overlay window. The routine is `XSolarisOvlCreateWindow`. It behaves exactly as `XCreateWindow` except that the resulting window will be an overlay window. The visual used to create the overlay can be any visual. These routines are described below. 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. The optimal pairs can be inquired 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.

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.

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 Transparency

Overlay windows are unique in that applications can render transparent paint to them. This can be done through a Solaris Visual graphics library by specifying in the Graphics Context (GC) for that library that the paint is to be transparent. Each library has a defined way of doing this which is described in subsequent sections.

Advanced Features of Overlay Windows

The following sections describe the characteristics of overlay windows that were not discussed above. They deal mostly with what makes an overlay window unique as a window. Some sections also deal with application portability issues.

Overlay Window Background

As defined in the X specification, windows can have a *background*. The main purpose of window background is so that something reasonable is displayed 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 a `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`.

A background of `None` will mean that no rendering will be performed when the overlay window encounters a condition that invokes background painting. Neither transparent nor opaque paint will be rendered.

When the background is a pixmap XID, the background will be rendered with opaque paint. The rendered pixel values will be derived from the pixmap as defined in the X specification.

When the background is a single pixel value, the background will be a solid color rendered with opaque paint.

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` will generate 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 will be 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 will be 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 will be 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” on page 72 and “Portability Inquiry Routines” on page 89 for more on the subject of pixel-sharing pairs.

If the pair is known to never 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.

Other Overlay Window Characteristics

In most respects, other than those listed above, an overlay window is just like a standard X `InputOutput` window.

Specifically:

- An overlay window can be mapped or unmapped. The routines `XMapWindow`, `XUnmapWindow`, `XMapSubwindows`, `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_propogate_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.

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 will receive 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, not 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 certain restricted 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.

7

Choosing Visuals

Multiple plane group (MPG) and single plane group (SPG) devices support the Solaris Visual Overlay Window API.

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 still provide the same type of overlay window but with slower rendering. Some devices can support overlays with a lot of 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 are described in further detail in the section “Portability Inquiry Routines” on page 89.

The assumption is made that each application has an ideal configuration of windows and colors that it would like to use. An application should start out by asking for the “best” overlay/underlay pair. Initially, the application should be quite bold in its definition of best—it should ask for its notion of the ideal 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 maybe it can abide

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. Applications are encouraged to use these routines to ensure portability to the widest range of graphics devices.

Interaction with Other Extensions

SHAPE

The shape of overlay windows can be controlled through the SHAPE extension just like a standard X window.

Multibuffering

Eventually applications will be able to use MBX to multibuffer overlay windows. However, the current version of Solaris does not support this feature. When it is supported, the `Xmbuf` functions will work on overlay windows just as they do on standard X windows.

Xlib Interface

This section contains the visual overlay type definitions. To use the routines described in this section:

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

XSolarisOvlPaintType

`XSolarisOvlPaintType` defines the paint type in each GC.

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

XSolarisOvlCreateWindow

`XSolarisOvlCreateWindow` is an X extension creation routine provided by `libXext.so`.

Synopsis

Create an overlay window.

```
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)
```

Arguments

The arguments for this routine are exactly the same as `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 it 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.

Description

This routine creates an overlay window with the given characteristics. It behaves exactly as its counterpart `XCreateWindow`, except the newly created window can be rendered into with both opaque and transparent paint, and the background is transparent.

XSolarisOvlIsOverlayWindow

`XSolarisOvlIsOverlayWindow` is an inquiry routine provided by `libXext.so`.

Synopsis

Indicates whether a given window is an overlay window.

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

Arguments

`display`
Specifies the connection to the X server.

`w`
Specifies the window.

Description

Returns `True` if the given window *w* is an overlay window. Otherwise returns `False`.

XSolarisOvlSetPaintType

`XSolarisOvlSetPaintType` is a paint type control routine provided by `libXext.so`.

Synopsis

Specifies the type of paint rendered by subsequent Xlib drawing with the given GC.

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

Arguments

`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.

Description

This routine controls the type of paint rendered by an Xlib GC. It controls whether Xlib drawing routines using this GC produce pixels on overlay windows that are opaque or transparent. 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.

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.

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 will be displayed.

By default, a GC renders opaque paint.

XSolarisOvlGetPaintType

`XSolarisOvlGetPaintType` is a paint type inquiry routine provided by `libXext.so`.

Synopsis

Get the current paint type set in the GC.

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

Arguments

`display`
Specifies the connection to the X server.

`gc`
The GC to be inquired from.

Description

This routine returns the current element of type `XSolarisOvlPaintType` associated with the given gc.

XSolarisOvlSetWindowTransparent

XSolarisOvlSetWindowTransparent is a window background control routine provided by `libXext.so`.

Synopsis

Sets the background state of an overlay window to be transparent.

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

Arguments

`display`
Specifies the connection to the X server.

`w`
The overlay window.

Description

This routine sets the background state of the given overlay to be transparent. Any background rendering that occurs after this request will cause the background to be transparent. Use `XChangeWindowAttributes()`, `XSetWindowBackground()`, or `XSetWindowBackgroundPixmap()` to change background state to any other value.

If w is not an overlay window, `BadMatch` is generated.

XSolarisOvlCopyPaintType

`XSolarisOvlCopyPaintType` is a rendering routine provided by `libXext.so`

Synopsis

Renders opaque and transparent paint into the destination drawable based on the paint type attributes of the pixels in the source drawable.

```
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)
```

Arguments

`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.

Description

This routine uses the paint type information of the specified rectangle of `src` to control a fill operation in the specified rectangle of `dst`. `src` and `dst` can be any type of drawable. If `src` 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 `src` is any other type of drawable, the bit-plane 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.

The following table 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 below the table.

Table 7-1 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.

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

If portions of the source rectangle are obscured or are outside the boundaries of the source drawable, the server generates exposure 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.

XSolarisOvlCopyAreaAndPaintType

`XSolarisOvlCopyAreaAndPaintType` is a rendering routine provided by `libext.so`.

Synopsis

Copies the given area and paint type data from one pair of drawables to another.

```
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)
```

Arguments

`display`
Specifies the connection to the X server.

`colorsrc`
The color information source drawable.

`painttypesrc`
The paint type information source drawable.

`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.

Description

This routine copies the specified area of `colorsrc` to the specified area of `colordst`. If `colordst` is not an overlay, it also fills the specified areas of `painttypedst` according to the paint type information specified in `painttypesrc`.

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

The following table 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 7-2 XSolarisOvlCopyAreaAndPaintType Possible 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.

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 2 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. Then call `XGetImage` on each of the drawables, to get the data from the server into the client's memory space.

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.

XReadScreen

`XReadScreen` is an image retrieval routing provided by `libext.so`.

Synopsis

Returns the displayed colors in a rectangle of the screen.

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

Arguments

`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.

Description

This routine 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 a picture of the what 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. It returns the color information from any window within the boundaries of the specified rectangle. Unlike `XGetImage`, the returned contents of visible regions of inferiors 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.

If `w` is an overlay window, the overlay color information is returned everywhere there is opaque paint in the specified rectangle. The color information of the underlay is returned everywhere 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) which contains transparent paint is the youngest non-inferior that has opaque paint at (x, y).

The color data is returned as an `XImage`. 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) will be 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.

Semantics of Existing Primitive Rendering Routines

All of the standard Xlib primitive rendering routines, such as `XDrawLines` and `XFillRectangles`, can be used to draw to overlay windows. In this case, the paint type attribute of the argument GC will be used to control the quality of the pixels rendered. The paint type attribute applies to both the foreground and background GC attributes.

This applies even to `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.

Semantics of Existing Pixel Transfer Routine

XGetImage

On non-overlay drawables, this 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 will be undefined. Clients who simply want to retrieve the display colors for a region on the screen should use `XReadScreen` instead.

XCopyArea and XCopyPlane

When both the source and destination drawables are non-overlay, these routine works as defined in the X11 specification.

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 will be 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 will be ignored and the destination pixels will be 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, `XSolarisOvlCopyAreaAndPaintType` should be used.

Portability Inquiry Routines

XSolarisOvlSelectPartner

`XSolarisOvlSelectPartner` is a visual inquiry routine provided by `libext.so`.

Synopsis

Given an underlay visual and a set of criteria, returns the overlay visual that best meets the criteria. Or, inversely, given an overlay visual and a set of criteria, returns the underlay visual that best meets the criteria.

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

Arguments

`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 only meaningful when the routine returns a value of `XSolarisOvlQualifiedSuccess`, or `XSolarisOvlCriteriaFailure`.

Argument Types

See the `XSolarisOvlSelectPartner` Description section for a full description of how these types should be used.

`XSolarisOvlSelectType`

An enumeration defining two types of selections that can be done in `XSolarisOvlSelectPartner`.

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

`XSolarisOvlVisualCriteria`
A structure defining various criteria to be used during visual selection, along with indications of the stringency of the criteria.

```
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)
```

These are described in the `XSolarisOvlSelectPartner` Description documentation that follows.

Return Types

`XSolarisOvlSelectStatus`
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.

Description

Portable applications using overlays may wish to 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 said to be *partners* of each other.

`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 in `pCriteria`. 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 which is not required.

The visual that matches all hard criteria and the most soft criteria is chosen. Its attributes are returned in `visinfoReturn`. 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.

`XSolarisOvlSelectPartner` supports a *degradation sequence* of criteria sets. This means that multiple criteria sets can be specified in a single call. First, an attempt is made to find a visual matching the first criteria set. If a visual is found which meets all of the hard criteria of the first set, this visual is chosen. If no visual met all hard criteria of the first set, a search is performed 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 which are less desirable can be specified in criteria sets following the first. 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 which follow, these data members are mentioned where applicable.

`XSolarisOvlVisualClass` specifies that the client desires 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 desires 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 it 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.

XSolarisOvlSelectPair

`XSolarisOvlSelectPair` is a visual inquiry routine provided by `libext.so`.

Synopsis

Given a set of criteria for both and overlay visual and underlay visual, selects an optimal overlay / underlay visual pair that best meets the criteria.

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

Arguments

`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 only meaningful 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 only meaningful when the routine returns a value of `XSolarisOvlQualifiedSuccess`, or `XSolarisOvlCriteriaFailure`.

Argument Types

See the Description section for a full description of how these types should be used.

`XSolarisOvlPairCriteria`
A structure defining various criteria to be used during visual selection, along with indications of the stringency of the criteria.

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

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

Return Types

`XSolarisOvlSelectStatus`

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

`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` contains 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` contains 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

Description

This 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 will be 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 will have an overlay that satisfies all the hard criteria specified for the overlay. The pair will have 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. It is implementation dependent which one is chosen.

Like `XSolarisOvlSelectPartner`, `XSolarisOvlSelectPair` supports a *degradation sequence* of criteria sets. This means that multiple criteria sets can be specified in a single call. First, an attempt is made to find a pair matching the first criteria set for both the overlay and the underlay. If a pair is found which meets all of the hard criteria of the first set, this pair is chosen. If no pair meets all hard criteria of the first set, a search is performed using the second criteria set. This process continues until either a pair is found that meets the 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 which are less desirable can be specified in criteria sets following the first. 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 the specification of `XSolarisOvlSelectPartner`.

Summary of New XLib Routines

This section lists the new routines and attributes defined by the Overlay Window API. They are all provided by `libXext.so`.

- `XSolarisOvlCreateWindow`
- `XSolarisOvlIsOverlayWindow`
- `XSolarisOvlSetPaintType`
- `XSolarisOvlGetPaintType`
- `XSolarisOvlCopyPaintType`
- `XSolarisOvlCopyAreaAndPaintType`

- `XReadScreen`
- `XSolarisOvlSelectPartner`
- `XSolarisOvlSelectPair`

Security Issues 8

OpenWindows 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. Otherwise, you do not need to change the default security configuration.

- Linked with a version of `Xlib` *previous* to OpenWindows Version 2 or X11R4. See "Host-Based" on page 102 for details.
- That is *statically* linked to OpenWindows Version 2 libraries *and* you want to use the SUN-DES-1 authorization protocol. See "SUN-DES-1" on page 103 for details.
- On a remote server. See "Running Clients Remotely, or Locally as Another User" on page 108 for details.

8

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 connection attempt logs to the server console as:

```
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 105 for more information.

User-Based

A user-based, or authorization-based mechanism allows you to explicitly give access 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, in which all users on that host can connect to the server. This is a weak form of access control; if that host has access to the server, all users on that host can connect to the server.

OpenWindows 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 (MIT) Massachusetts Institute of Technology. A *magic cookie* is a long, randomly generated binary password. At server start-up, 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 servers' 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 RPC (Remote Procedure Call) and requires DES (Data Encryption Software) support. (See the *Network Interfaces Programmer's Guide* for more information). 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 107 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 OpenWindows 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 that 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), 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.

There are two programs that can be used to change a server’s access control mechanism: `xhost` and `xauth`. For more information, see these `man` pages. 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 are starting 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 102 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 another user can merge the data into their `.Xauthority` file, thus allowing them access to your server, or to the server in which you connect.

See “Allowing Access When Using MIT-MAGIC-COOKIE-1” on page 107 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 this example, to allow new user *somebody* to run on *myhost*:

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

2. **The new user must use** `xauth` **to add the entry into their** `.Xauthority` **file.**
   For this example, 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 in 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 107 or "Allowing Access When Using SUN-DES-1" on page 107.

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 will be displayed on the remote machine, *remotehost*.

```
myhost% client_program&
```

Multi-Buffering Application Program Interface, Version 3.2

A

This appendix describes the C language API (application program interface) to the MBX (Multi–Buffering) extension.[1] These routines provide direct access to the protocol and add no additional 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 "MBX (Multi-Buffering) Extension" on page 22 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.

A

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

- `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 – The window mode constant `MultibufferModeStereo`, and the window side constants `MultibufferSideMono`, `MultibufferSideLeft`, and `MultibufferSideRight` are not supported in SunSoft's MBX implementation.

Event Mask Constants

- `MultibufferClobberNotifyMask`
- `MultibufferUpdateNotifyMask`

Valuemask Constants

- `MultibufferWindowUpdateHint`
- `MultibufferBufferEventMask`

Clobber State Constants

- `MultibufferUnclobbered`

- `MultibufferPartiallyClobbered`
- `MultibufferFullyClobbered`

New Structures

Several new structure types are defined. Most structures are introduced in the function discussion of a function that requires it as a parameter. The following structures are not parameters in any of the functions discussed in "MBX Functions" on page 113.

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 desired.

update_action
Specifies the update action to be applied to the buffers. See *Update action constants* above for allowable values.

update_hint
Specifies the update hint for the buffers. See *“Update Hint Constants” on page 111* 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 above */
    int update_hint;/* see Update hint constants above */
    int window_mode;/* see Window mode constants above */
    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 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 multibuffering 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.

A

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.

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. There are several different types of default visuals: built-in, server, and allowable.

Depth

The depth is the pixel size.

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.

Dumb Frame Buffer

A display device that consists of display memory only.

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.

Express Interest

A client that has specifically asked to be informed of an event has expressed interest in that event.

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 and GS provide visuals of different depths; they are referred to as MPG or multi-depth devices.

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.

Render

To draw; to cause a graphics device to draw on its display.

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.

Server Default Colormap

Unless specified otherwise, a PseudoColor visual is created when the server starts up. This colormap is the server default colormap.

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. The visual classes supported by OpenWindows are: PseudoColor, StaticColor, GrayScale, StaticGray, TrueColor, and DirectColor.

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 OpenWindows 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 OpenWindows server handles all communication with the window manager.

Index

Symbols

.Xauthority file 105–106

A

Adobe FTP site 33
Adobe public access file server 33
allowable default visual 55
API (application programmer's interface) 4
architecture 1–2
architecture drawing 2
authorization protocols, *See* security
authorization-based access control mechanism, *See* security

B

bdftopcf 36
bdftosnf 37
bitmap distribution format 36
bitmap fonts 36–37, 43–44
Black pixel location note 25
built-in default visual 55
bus, definition of 52
bus, used in SPARCsystems 52
BW2 display device, description of 52
  See also display devices

C

CG3 display device, description of 53
  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 108
  running remotely 108
client library
  for DPS 28
color
  color name database 24
  recommendations 24
colormaps
  demo
    content confusion note 59
    location of (xcolor) 58
    multiple LUT note 59

hardware, multiple 58
installation
InstallColormap request 58
UninstallColormap request 58
window manager policy 58
compose key support 24
compressing font files 37
contexts
and DPS 30
secure 32
three ways to share VM 32

D

defclass option 56
defdepth option 56
demonstration applications, location of 4
DES (Data Encryption Software), with SUN-DES-1 103
device driver, definition of 52
device name, definition of 52
device, *See* display devices
directory structure 7
display devices
definition of 49
bus, definition of 52
bus, used in Sun SPARCsystems 52
BW2
description of 52
support information 51
CG3
description of 53
CG6, *See* GX and GXplus
CG8, *See* TC
default visual assumptions 57
device driver, definition of 52
device name, definition of 52
GT
window damage note 53
GX
description of 53
GXplus
description of 53
support information 51
treated as GX note 53
product name, definition of 51
programming hints 57
supported devices table 51
TC
description of 53
DISPLAY environment variable 108
DPS
.upr files 40
introduction 27–28
applications modifed to use DPS 31
changing the resource path 40–41
client library 28
extension 28–30
font enhancements 30
how applications use 28
libraries supported 31
PostScript interpreter 28
pswrap translator 28
security issues 32–33

F

F3 fonts 38
character set supported by 46
F3BitMap resource 37
font management library, definition of 13
fonts
.afm file 42

.enc file 42, 47
.map file 42
.ps file 42
.trans file 42
.upr file 43, 45
adding bitmap fonts 43, 44
adding new fonts 43–47
adding scalable fonts 44, 47
and X terminals 47
available in X11 and DPS 42
default font path in X11 39
directory structure 38–39
file suffixes 42
files included in openwindows 42–43
fonts.alias file 44, 46
fonts.dir file 43
fonts.scale file 45
formats 35–36
outline and bitmap 36, 37
replacing outline with bitmap 37
using F3 fonts 38
ftp program 22
ftp, accessing Adobe FTP site 33

G

grayvis option 56
GX display device, description of 53
See also display devices
GXplus display device, description of 53
See also display devices

H

host-based access control mechanism, *See* security

I

ICCCM (Inter-Client Communication Conventions Manual)
definition of 21
compliance 21
InstallColormap request 58

L

libraries
DPS. list of 31
X, list of 19

M

makebdf 36
makepsres 44
MBX (multi-buffering) X extension 22
implementation caution note 22
MIT-MAGIC-COOKIE-1 authorization protocol, *See* security
MIT-SHM (Shared Memory) X extension 23
MIT-SUNDRY-NONSTANDARD X Extension 23
mkfontdir 43
multiple hardware colormap 58
multiple plane group, characteristics of 50

N

NISdomainname, definition of 106

O

OLIT (OPEN LOOK Intrinsics Toolkit) 5
OPEN LOOK Graphical User Interface 3
OPEN LOOK Window Manager (olwm) 6
openwin command

-defclass option 56
-defdepth option 56
-grayvis option 56
-noauth option 102, 104
outline fonts 36, 37
overlay windows
advanced features 67–73
and multibuffering 73
and shape extension 73
background 67
backing store 69
border 68
choosing visuals 72–73
colormap 69
gravity 69
input distribution model 70
interaction with other extensions 73
other characteristics 69
print capture 71
and existing pixel transfer routines 89
and existing primitive rendering routines 88
basic features 65–67
creation 66
definition 65–66
rendering transparency 67
viewability 67
new Xlib routines, summary 99
portability inquiry routines 89–99
Xlib Interface 73–88

P

portable compiled format 36
compressed files 37
PostScript interpreter 28
product overview 1–2
pswrap translator 28

R

resource files (.upr) 40
RPC (Remote Procedure Call), with SUN-DES-1 103

S

scalable fonts 44–47
screen
default visual 55
secure context creation 32
security 101–108
.Xauthority file 105–106, 107
contents with MIT-MAGIC-COOKIE-1 106
contents with SUN-DES-1 106
access control mechanisms 102–103
definition of 102
how both are active 105
authorization protocols 103–104
default configuration 103
default, how to change 104
authorization-based, *See* user-based clients
running locally as another user 108
running remotely 108
connection attempt error message 102
default configuration 103
determining if configuration change is required 101
host-based, backward compatibility 102
host-based, definition of 102

MIT-MAGIC-COOKIE-1
authorization protocol 103
NISdomainname, definition of 106
-noauth option 102
weakens security warning 104
server
manipulating access 105–108
allowing access with MIT-MAGIC-COOKIE-1 107
allowing access with SUN-DES-1 107
SUN-DES-1 authorization protocol
definition of 103–104
need to reconfigure 101
user-based, definition of 102
userid, definition of 106
xauth program 105, 107
xhost program 105, 107
server
applications that run with 14
architecture 12
architecture diagram 12
changing X11 default font path 39–40
DDX layer, definition of 13
DIX layer, definition of 13
font management library, definition of 13
manipulating access control 105–108
OS layer, definition of 13
server natural format 37
SHAPE extension 73
SHAPE X extension 23
staticvis option 56, 57
SUN-DES-1 authorization protocol, *See* security
system file access 32

T

TC display device, description of 53
See also display devices
toolkit 4
OLIT (OPEN LOOK Intrinsics Toolkit) 5
XView 5

U

UninstallColormap request 58
user-based access control mechanism, *See* security

V

virtual memory 30
visuals
default 55–57
allowable 55
assumptions 57
built-in 55
defclass option 56
defdepth option 56
get with XGetVisualInfo function 55
grayvis option 56
screen 55
staticvis option 56, 57
troubleshooting / error messages when changing 57
warning using XGetVisualInfo with grayvis 56
gamma-corrected 59–64
multiple plane group, characteristics of 50
VM (virtual memory) 30, 32

shared 30

W

White pixel location note 25
window manager colormap installation policy 58

X

X
applications supported 20
compose key 24
extensions
how to access standards 22
MBX (multi-buffering) 22
implementation caution note 22
MIT-SHM (Shared Memory) 23
MIT-SUNDRY-NONSTANDARD 23
SHAPE 23
XInput 22
XTEST 23
features 19–21
features not supported 21
libraries supported 19
library (Xlib), definition of 18
See also Xlib
protocol
definition of 18
terminals and fonts 47
toolkits 19
X11R5 compliance 17
X Consortium
description of 17
extensions supported 13, 21
X Window System
overview 17
X11R5 compliance 17
xauth program 105, 107
xcolor colormap demo 58
xcolor colormap demo, multiple LUT note 59
XCopyArea, and overlay windows 89
XCopyPlane, and overlay windows 89
XDPSCreateSecureContext 33
XGetImage, and overlay windows 89
XGetVisualInfo function
grayvis option warning 56
list default visual 55
xhost program 105, 107
XInput X Extension 22
XLFD
font naming convention 46
Xlib
definition of 18
xlsfonts 44, 47
XOvlCopyAreaAndPaintType 81
XOvlCopyPaintType 79
XOvlCreateWindow 74
XOvlGetPaintType 77
XOvlIsOverlayWindow 75
XOvlPaintType 74
XOvlSelectPair 96
XOvlSelectPartner 89
XOvlSetPaintType 76
XOvlSetWindowTransparent 78
XReadScreen 86
XSetFontPath 40
XTEST X Extension 23
XView 5