VMS

digital

VMS DECwindows Guide to Xlib Programming: MIT C Binding

Order Number: AA–MG24A–TE

VMS DECwindows Guide to Xlib Programming: MIT C Binding

Order Number: AA–MG24A–TE

December 1988

This manual is a guide to programming Xlib routines.

Revision/Update Information:	This is a new manual.
Software Version:	VMS Version 5.1

digital equipment corporation
maynard, massachusetts

December 1988

ZK4733

Production Note

This book was produced with the VAX DOCUMENT electronic publishing system, a software tool developed and sold by DIGITAL. In this system, writers use an ASCII text editor to create source files containing text and English-like code; this code labels the structural elements of the document, such as chapters, paragraphs, and tables. The VAX DOCUMENT software, which runs on the VMS operating system, interprets the code to format the text, generate a table of contents and index, and paginate the entire document. Writers can print the document on the terminal or line printer, or they can use DIGITAL-supported devices, such as the LN03 laser printer and PostScript printers (PrintServer 40 or LN03R ScriptPrinter), to produce a typeset-quality copy containing integrated graphics.

Contents

Preface

This manual describes how to program Xlib routines using the MIT C binding. VMS DECwindows includes the MIT binding for Xlib programmers using the C programming language and other languages that support pointers.

The manual includes an overview of Xlib and tutorials that show how to use Xlib routines.

Intended Audience

This manual is intended for experienced programmers who need to learn graphics programming using Xlib routines. Readers should be familiar with a high-level language. The manual requires minimal knowledge of graphics programming.

Document Structure

This manual is organized as follows:

- Chapter 1 provides an overview of Xlib, a sample Xlib program, and a guide to debugging Xlib programs.
- Chapters 2 through 9 provide tutorials that show how to use Xlib routines and include descriptions of predefined Xlib data structures and code examples that illustrate the concepts described.

This manual also includes the following appendixes:

- Appendix A is a guide to using the VMS DECwindows font compiler.
- Appendix B lists routines that require Xlib to issue protocol requests to the server.
- Appendix C lists the VMS DECwindows named colors.
- Appendix D lists VMS DECwindows fonts.

Associated Documents

The following documents contain additional information:

- *VMS DECwindows Guide to Application Programming*—Provides an overview of programming in the VMS DECwindows environment and a guide to programming the XUI Toolkit
- *VMS DECwindows Xlib Routines Reference Manual*—Provides detailed descriptions of each Xlib routine
- *XUI Style Guide*—Describes the standard XUI user interface

Conventions

The following conventions are used in this manual:

mouse	The term *mouse* is used to refer to any pointing device, such as a mouse, a puck, or a stylus.
MB1, MB2, MB3	MB1 indicates the left mouse button, MB2 indicates the middle mouse button, and MB3 indicates the right mouse button. (The buttons can be redefined by the user.)
. . .	A vertical ellipsis indicates the omission of items from a code example or command format; the items are omitted because they are not important to the topic being discussed.
[]	In format descriptions, brackets indicate that whatever is enclosed is optional; you can select none, one or all of the choices.
boldface text	Boldface text represents the introduction of a new term or the name of an argument, a constant, or a flag.
italic text	Italic text represents a variable or a client-defined routine.
UPPERCASE TEXT	Uppercase letters indicate the name of a routine or a system service call.

1 Programming Overview of Xlib

The VMS DECwindows programming environment includes Xlib, a library of low-level routines that enable the VMS DECwindows programmer to perform windowing and graphics operations.

This chapter provides the following:

- An overview of the library
- A description of error handling conditions
- Xlib debugging techniques

Additionally, the chapter includes an introductory Xlib program. The program includes annotations that are explained more completely in the programming descriptions in later chapters of this guide.

1.1 Overview of Xlib

The VMS DECwindows programming environment enables application programs, called **clients**, to interact with workstations using the X Window System, Version 11 protocol. The program that controls workstation devices such as screens and pointing devices is the **server**. Xlib is a library of routines that enables a client to communicate with the server to create and manage the following:

- Connections between clients and the server
- Windows
- Colors
- Graphics characteristics such as line width and line style
- Graphics
- Cursors
- Fonts and text
- Pixmaps and offscreen images
- Windowing and sending graphics between clients
- Client notification of windowing and graphics operations

Xlib processes some client requests, such as requests to measure the width of a character string, within the Xlib library. It sends other client requests, such as those pertaining to putting graphics on a screen or receiving device input, to the server.

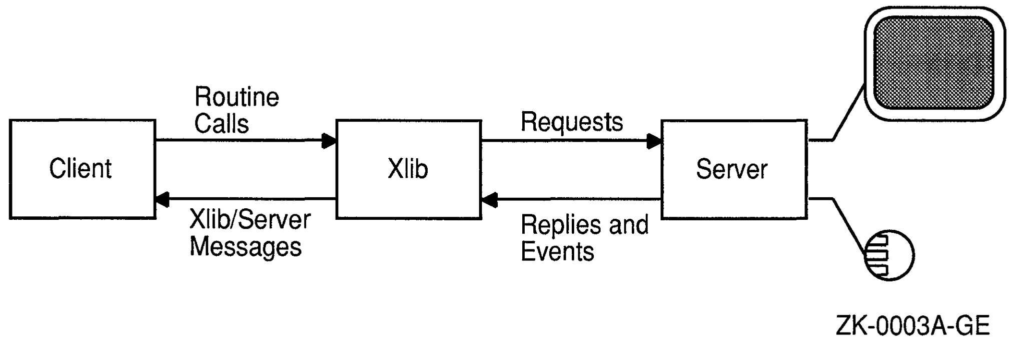
The server returns information to clients through either replies or events. Replies and events both return information to clients; the server returns replies synchronously and events asynchronously.

Appendix B lists routines that cause Xlib to send requests to the server.

Figure 1–1 illustrates the relationships among client, Xlib, and server. The client calls Xlib routines, which always reside on the client system. If possible, Xlib processes calls internally and returns information to the client when appropriate. When an Xlib function requires server intervention, Xlib generates a request and sends the request to the server.

The server may or may not reside on the same system as the client and Xlib. In either case, Xlib communicates with the server through a transport protocol, which can be either local shared memory or DECnet.

Figure 1–1 Client, Xlib, and Server



1.2 Sample Xlib Program

The introductory Xlib program described in Example 1–1 illustrates the structure of a typical client program that uses Xlib windowing and graphic operations. The program creates two windows, draws text in one of them, and exits if the user clicks any mouse button while the cursor is in the window containing text.

The main loop of the program comprises two client-defined routines: *doInitialize* and *doHandleEvents*.

This section describes these routines and introduces fundamental concepts about Xlib resources, windowing, and event-handling.

1.2.1 Sample Initialization Routine

The sample program begins by calling a client-defined routine, *doInitialize*. The routine creates the resources the client needs to perform tasks. Xlib resources include windows, fonts, pixmaps, cursors, color maps, and data structures that define the characteristics of graphics objects. The sample program uses a default font, default cursor, default color map, client-defined windows, and a client-defined data structure that specifies the characteristics of the text displayed.

The *doInitialize* routine makes a connection between the client and the server. The client-server connection is called the **display**. After making the connection, or opening the display, the client can get display information from the server. For example, immediately after opening the display, the program calls the DEFAULT SCREEN OF DISPLAY routine to get the identifier of the default screen. The program uses the identifier as an argument in a variety of routines it calls later.

1.2.1.1 Creating Windows

A window is an area of the screen that either receives input or both receives input and displays graphics.

Windows in the X Window System are hierarchically related. At the base of the hierarchy is the **root window**. All windows that a client creates after opening a display are **inferiors** of the root window. The sample program includes two inferiors of the root window. First generation inferiors of a window are its **children**. The root window has one child, identified in the sample as *window1*. The window named *window2* is an inferior of the root window and a child of *window1*.

To complete the window genealogy, all windows created before a specified window and hierarchically related to it are its ancestors. In the sample program, *window1* has one ancestor (the root window); *window2* has two ancestors (the root window and *window1*).

1.2.1.2 Defining Colors

Defining background and foreground colors is part of the process of creating windows in the sample program. The *doDefineColor* routine allocates named VMS DECwindows colors for client use in a way that permits other clients to share the same color resource. For example, the routine specifies the VMS DECwindows color named "light grey" as the background color of *window2*. If other clients were using VMS DECwindows color resources, they too could access the VMS DECwindows data structure that defines "light grey." Sharing enables clients to use color resources efficiently.

The sample program calls the *doDefineColor* routine again in the next step of initialization, creating the graphics context that defines the characteristics of a graphics object. In this case, the program defines foreground and background colors used when writing text.

1.2.1.3 Working with the Window Manager

Most clients run on systems that have a window manager, which is an Xlib application that controls conflicts between clients. Clients provide the window manager with information about how it should treat client resources, although the manager can ignore the information. The sample program provides the window manager with information about the size and placement of *window1*. Additionally, the program assigns a name that the window manager displays in the title bar of *window1*.

1.2.1.4 Making Windows Visible on the Screen

Creating windows does not make them visible on the screen. To make its windows visible, a client must **map** them, painting the windows on a specified screen. The last step of initializing the sample program is to map *window1* and *window2*.

1.2.2 Sample Event-Handling Routine

The core of an Xlib program is a loop in which the client waits for the server to notify it of an **event**, which is a report of either a change in the state of a device or the execution of a routine call by another client. The server can report 30 types of events associated with the following occurrences:

- Key presses and releases
- Pointer motion
- Window entries and exits
- Changes of keyboards receiving input
- Changes in keyboard configuration
- Window and graphics exposures
- Changes in window hierarchy and configuration
- Requests by other clients to change windows
- Changes in available color resources
- Communication from other clients

When an event occurs, the server sends information about the event to Xlib. Xlib stores the information in a data structure. If the client has specified an interest in that kind of event, Xlib puts the data structure on an event queue. The *doHandleEvents* routine polls the event queue to determine if it contains an event of interest to the client. When the routine finds an event that is of interest to the client, the *doHandleEvents* routine calls one or more other routines.

Because Xlib clients do their essential work in response to events, they are considered event driven.

The sample program continually checks its event queue to determine if a window has been made visible or a button has been clicked. When the server informs it of either kind of event, the program performs its real work, as follows.

If the event is a window exposure, the program calls the *doExpose* routine. This routine checks to determine whether or not the window exposed is *window2* and the event is the first instance of the exposure. If both conditions are true, the program writes a message into the window.

If the event is a button press, the program calls the *doButtonPress* routine. This routine checks to make certain the cursor was in *window2* when the user clicked the mouse button. If the user clicked the mouse button when the cursor was on the root window or *window1*, the program reminds the user to click on *window2*. Otherwise, the program initiates a series of shutdown routines.

The shutdown routines unmap *window1* and *window2*, free resources allocated for the windows, break the connection between the sample program and its server, and exit the system. On the VMS operating system, clients only need to call SYS$EXIT. Exiting the system causes the other shutdown operations to occur. The call to SYS$EXIT breaks the connection between client and server, which frees resources allocated for client windows, and so forth.

See Example 1–1 for the sample Xlib program.

Example 1–1 Sample Xlib Program

```
#include <decw$include/Xlib.h>
#include <decw$include/Xutil.h>

#define FontName "-ADOBE-NEW CENTURY SCHOOLBOOK-MEDIUM-R-NORMAL--*-140-*-*-P-*"
#define WindowName "Sample Xlib Program"

Display *dpy;
Window window1,window2;
GC gc;
Screen *screen;
int n, state = 0;
char *message[]= {
    "Click here to exit",
    "Click HERE to exit!"
    };

static void doInitialize( );
static int doDefineColor( );
static void doCreateWindows( );
static void doCreateGraphicsContext( );
static void doLoadFont( );
static void doExpose( );
static void doWMHints( );
static void doMapWindows( );
static void doHandleEvents( );
static void doButtonPress( );

/********************** The main program ******************************/

static int main()
{
    doInitialize( );
    doHandleEvents( );
}
```

(continued on next page)

Example 1–1 (Cont.) Sample Xlib Program

```
/***************** doInitialize *************************/
static void doInitialize( )
{
❶  dpy = XOpenDisplay(0);
    if (!dpy){
        printf("Display not opened!\n");
        exit(-1);
    }
    screen = XDefaultScreenOfDisplay(dpy);

❷  XSynchronize(dpy,1);

    doCreateWindows( );

    doCreateGraphicsContext( );

    doLoadFont( );

    doWMHints( );

    doMapWindows( );
}

/******* doCreateWindows *********/
❸static void doCreateWindows( )
{
    int window1W = 400;
    int window1H = 300;
    int window1X = (XWidthOfScreen(screen)-window1W)>>1;
    int window1Y = (XHeightOfScreen(screen)-window1H)>>1;
    int window2X = 50;
    int window2Y = 75;
    int window2W = 300;
    int window2H = 150;
    XSetWindowAttributes xswa;

    /* Create the window1 window */

    xswa.event_mask = ExposureMask | ButtonPressMask;
    xswa.background_pixel = doDefineColor(1);

    window1 = XCreateWindow(dpy, XRootWindowOfScreen(screen),
        window1X, window1Y, window1W, window1H, 0,
        XDefaultDepthOfScreen(screen), InputOutput,
        XDefaultVisualOfScreen(screen), CWEventMask | CWBackPixel, xswa);

    /* Create the window2 */

    xswa.event_mask = ExposureMask | ButtonPressMask;
    xswa.background_pixel = doDefineColor(2);

    window2 = XCreateWindow(dpy, window1, window2X, window2Y, window2W,
        window2H, 4, XDefaultDepthOfScreen(screen), InputOutput,
        XDefaultVisualOfScreen(screen), CWEventMask | CWBackPixel, xswa);
}

/******** Create the graphics context *********/
❹static void doCreateGraphicsContext( )
{
    XGCValues xgcv;

    /* Create graphics context. */
```

(continued on next page)

Example 1–1 (Cont.) Sample Xlib Program

```
    xgcv.foreground = doDefineColor(3);
    xgcv.background = doDefineColor(2);

    gc = XCreateGC(dpy, window2, GCForeground | GCBackground, xgcv);
}

/******* Load the font for text writing ******/
❺static void doLoadFont( )
{
    Font font;

    font = XLoadFont(dpy, FontName);
    XSetFont(dpy, gc, font);
}

/******* Create color ***********************/
❻static int doDefineColor(n)
{
    int pixel;
    XColor exact_color,screen_color;
    char *colors[] = {
        "dark slate blue",
        "light grey",
        "firebrick"
        };

    if ((XDefaultVisualOfScreen(screen))->class == PseudoColor
        || (XDefaultVisualOfScreen(screen))->class == DirectColor)
        if (XAllocNamedColor(dpy, XDefaultColormapOfScreen(screen),
           colors[n-1], &screen_color, &exact_color))
               return screen_color.pixel;
     else
        switch (n) {
            case 1:             return XBlackPixelOfScreen(screen); break;
            case 2:             return XWhitePixelOfScreen(screen); break;
            case 3:             return XBlackPixelOfScreen(screen); break;
        }
}

/******** do WMHints *************/
❼static void doWMHints( )
{
    XSizeHints xsh;

    /* Define the size and name of the window window1 */

    xsh.x = 362;
    xsh.y = 282;
    xsh.width = 400;
    xsh.height = 300;
    xsh.flags = PPosition | PSize;

    XSetNormalHints(dpy, window1, &xsh);

    XStoreName(dpy, window1, WindowName);
}
```

(continued on next page)

Example 1–1 (Cont.) Sample Xlib Program

```
/******** doMapWindows ***********/
❽static void doMapWindows( )
{
    XMapWindow(dpy, window1);
    XMapWindow(dpy, window2);
}

/****************** doHandleEvents **********************/
❾static void doHandleEvents( )
{
    XEvent event;
    for ( ; ; ) {
        XNextEvent(dpy, &event);
        switch (event.type) {
            case Expose:                doExpose(&event); break;
            case ButtonPress:           doButtonPress(&event); break;
        }
    }
}

/***** Write the message in the window *****/
static void doExpose(eventP)
XEvent *eventP;
{
    /* If this is an expose event on our child window, then write the text. */

    if (eventP->xexpose.window != window2) return;
    XClearWindow(dpy, window2);
    XDrawImageString(dpy, window2, gc, 75, 75, message[state],
        strlen(message[state]));
}

/***************** doShutdown ***************************/
static void doButtonPress(eventP)
XEvent *eventP;
{
    if (eventP->xexpose.window != window2) {
        state = 1;
        XDrawImageString(dpy, window2, gc, 75, 75, message[state],
            strlen(message[state]));
        return;
    }

    /* Unmap and destroy windows */

❿  XUnmapWindow(dpy, window1);
    XDestroyWindow(dpy, window1);

    XCloseDisplay(dpy);

    sys$exit (1);
}
```

❶ For information about connecting client and server, see Chapter 2.

❷ Xlib buffers client requests and sends them to the server asynchronously. This causes clients to receive errors after they have occurred. When debugging a program, call the SYNCHRONIZE routine to enable synchronous error reporting. Using the

SYNCHRONIZE routine has a serious negative effect on performance. Clients should call the routine only when debugging. For more information about debugging, see Section 1.4.

❸ For information about creating windows, see Chapter 3.

❹ Before drawing a graphics object on the screen, clients must define the characteristics of the object. The *doCreateGraphicsContext* routine defines the foreground and background values for writing text. For information about defining graphics characteristics, see Chapter 4.

❺ The sample program loads a VMS DECwindows font, New Century Schoolbook Roman 14, which the program uses to write the text in *window2*. For information about loading fonts, see Chapter 8.

❻ VMS DECwindows includes named colors for the convenience of clients. The sample program uses the named colors "dark slate blue," "light grey," and "firebrick." It shares the named colors it uses with other clients. For information about sharing colors, whether named or client-defined, see Chapter 5. For information about defining colors for exclusive use, see Section 5.4. For a list of named VMS DECwindows colors, see Appendix C.

❼ For more information about window management, see Section 3.5.1.

❽ Mapping windows makes them visible on the screen. For information about window mapping, see Chapter 3

❾ For more information about event handling, see Chapter 9.

❿ When a client exits a VMS DECwindows program on the VMS operating system, the series of calls to unmap and destroy windows and close the display occurs automatically.

1.3 Handling Error Conditions

Xlib differs from most VMS programming libraries in the way it handles error conditions. In particular, Xlib does not perform any validation of input arguments when an Xlib routine is called.

If the input arguments are incorrect, the server usually generates an error event when it receives the Xlib request. Unless the client has specified an error handler, the server invokes the default Xlib error handler, which prints out a diagnostic message and exits. For more information about the Xlib error handler, refer to Section 9.13.2.

In some cases, Xlib signals a fatal access violation (SYS-F-ACCVIO) when passed incorrect arguments. This occurs when arguments are missing or are passed using the wrong addressing mode (passed by value instead of passed by reference).

1.4 Debugging Xlib Programs

As noted in Section 1.1, Xlib handles client requests asynchronously. Instead of dispatching requests as it receives them, Xlib buffers requests to increase communication efficiency.

Buffering contributes to delays in error reporting. Asynchronous reporting enables Xlib and the server to continue processing client requests despite the occurrence of errors. However, buffering contributes to the delay between the occurrence and client notification of an error.

As a result, programmers who want to step through routines to locate errors must override the buffering that causes asynchronous communication between client and server. To override buffering, use the SYNCHRONIZE routine. Example 1–1 includes a SYNCHRONIZE call as a debugging tool. Use the SYNC routine if you are interested in a specific call. The SYNC routine flushes the output buffer and then waits until all requests have been processed.

2 Managing the Client-Server Connection

A client requires one or more servers to process requests and return keyboard and mouse input. The server can be located either on the same system as the client or at a remote location where it is accessed across a network.

This chapter describes the following topics related to managing the client-server connection:

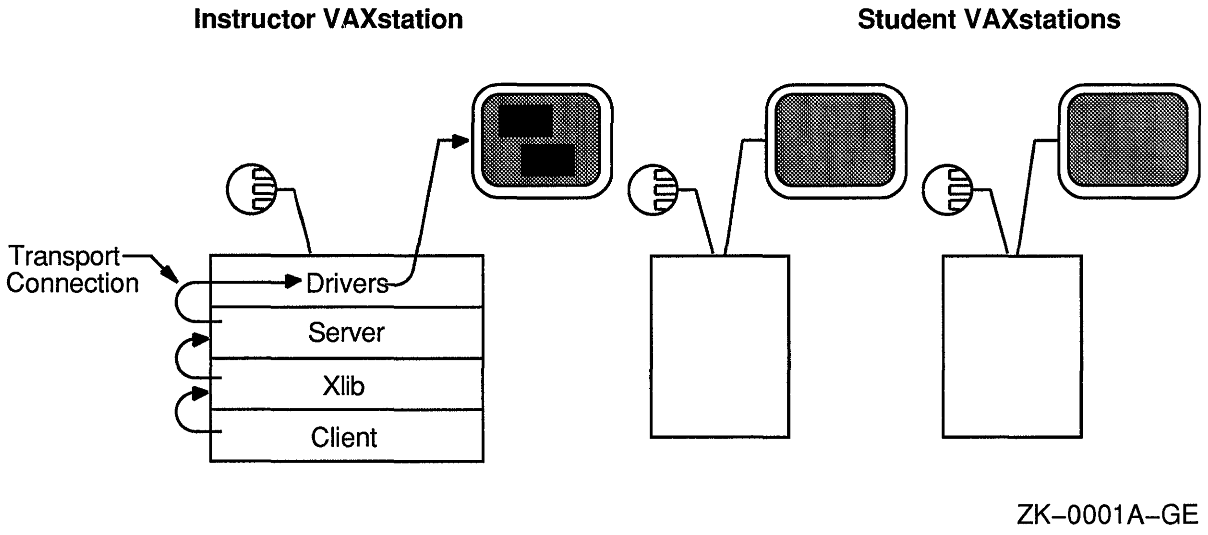
- Overview of the client-server connection
- Opening and closing a display
- Getting information about a display
- Managing sending requests to the server

2.1 Overview of the Client-Server Connection

A client using Xlib makes its first call to open a display. After opening a display, the client can get display information from and send requests to the server. To increase the efficiency of the client-server connection, Xlib buffers client requests.

To understand the relationship between a display and hardware, consider the classroom illustrated in Figure 2–1. The server and an instructor client program are running on the instructor VAXstation, which includes a screen, a keyboard, and a mouse. When the instructor opens a display, Xlib establishes a connection between the instructor client program and the server. The instructor can output graphics on the instructor VAXstation screen.

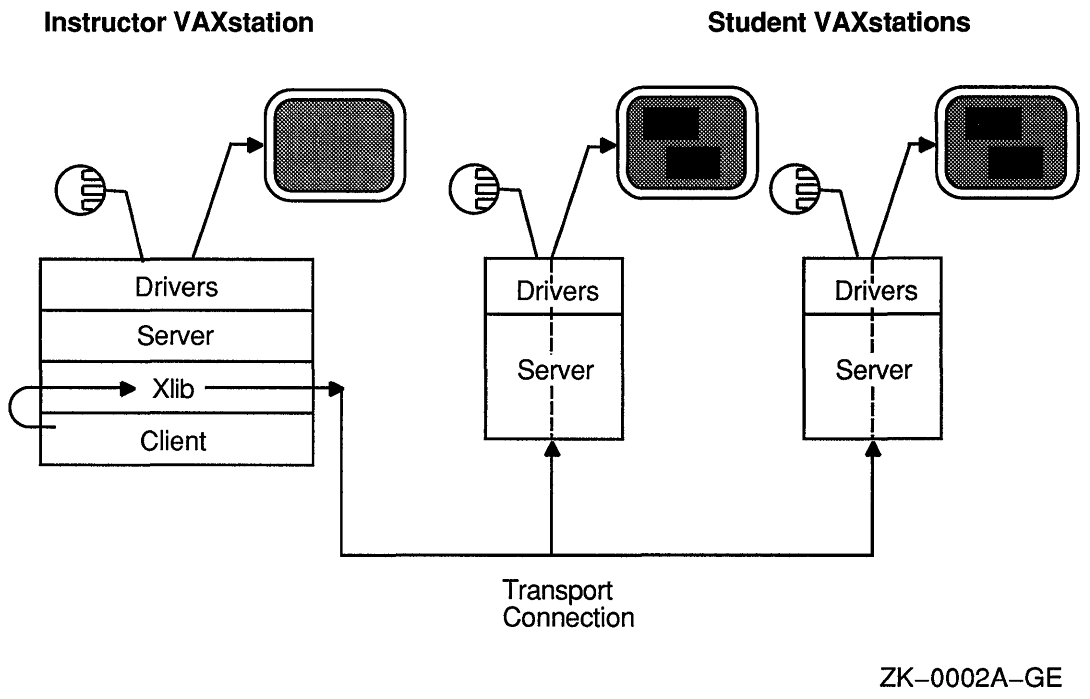
Figure 2–1 Graphics Output to Instructor VAXstation



If the instructor wants to output graphics to student screens, each student VAXstation must be running a server, and the client program must be connected to each server, as Figure 2–2 illustrates. Unlike the prior example, where the client program opened one display by making an internal connection with the server running on the VAXstation, here the client program establishes connections with multiple servers.

Xlib also enables multiple clients to establish connections with one server. For example, to output student work on the instructor screen, each student must open a display with the server running on the instructor VAXstation.

Figure 2–2 Graphics Output to Student VAXstations



2.2 Establishing the Client-Server Connection

The OPEN DISPLAY routine establishes a connection between the client and the server. The OPEN DISPLAY routine call has the following format:

```
display=XOpenDisplay(display_name)
```

In this call, **display_name** is a string that specifies the node on which the server is running and the transport mechanism used to make the connection between the client and the server. If the transport mechanism is local shared memory, users should use the DCL command SET DISPLAY to define which display to open and pass a null argument to the OPEN DISPLAY routine. The null argument causes the server to search for the definition of the display. If the transport mechanism is DECnet, the **display_name** argument has the following format:

```
hostname::number.screen
```

The elements of the argument are as follows:

Elements	Description
hostname	The host on which the server is running. The double colons indicate that the transport mechanism is DECnet.
number	The number of the display on the host machine. If the client and server are physically running in the same CPU, clients can specify a display number of zero, which causes the transport to use a version of DECnet that optimizes local performance.
screen	The screen on which client input and output is handled.

See Example 1–1 for an example of defining a display.

If successful, OPEN DISPLAY returns a unique identifier of the display.

Refer to the *VMS DECwindows User's Guide* for more information about specifying a display.

2.3 Closing the Client-Server Connection

Although Xlib automatically destroys windows and resources related to a process when the process exits the server, clients should close their connection with a server explicitly. Clients can close the connection using the CLOSE DISPLAY routine. CLOSE DISPLAY destroys all windows associated with the display and all resources the client has allocated. The CLOSE DISPLAY routine call has the following format:

```
XCloseDisplay(display)
```

For an example of closing a display, see Example 1–1.

After closing a display, clients should not refer to windows, identifiers, and other resources associated with that display.

When a display is closed automatically or by an explicit call to CLOSE DISPLAY, the server does the following:

- Discards all input events selected by the client. For information about input events, see Chapter 9.
- If the client has marked the keyboard, specific keys, the pointer button, the pointer, or the server for its exclusive use, the server releases them for use by other clients.
- Determines what happens to client resources after the display is closed.

If the server is to destroy all client resources, it destroys them as follows:

- Examines each window in the client **save set**. The save set is a list of windows that other clients are using. If a window is a member of the save set, the server reparents the window to an ancestor not created by the client.
- Maps the save set window, if it is unmapped. The server does this even if the save set window was not a subwindow of a window created by the client.
- Destroys all windows created by the client after examining each in the client save set.

- Frees each nonwindow resource (font, pixmap, cursor, color map, and graphics context) created by the client.
- Frees all colors and color map entries allocated by the client.

When the last connection to the server closes and the server is to destroy all client resources, the server performs the following additional steps:

- Resets its state as if it had just been started
- Deletes all identifiers except predefined names of window characteristics
- Deletes all information associated with the root window
- Resets all device maps and attributes (key click, bell volume, acceleration) and the **server access control list**, a list of hosts that can run client programs
- Restores the standard cursors and root **tile**, which is a pixmap replicated to create a window background
- Restores the default font path
- Restores input focus to the root window

The server does not perform reset operations if a client requests the server to retain its resources.

Refer to the *VMS DECwindows Xlib Routines Reference Manual* for information about the SET CLOSE DOWN MODE routine.

2.4 Getting Information About the Client-Server Connection

After opening a display, clients can get information about the client-server connection using routines listed in Table 2–1. Clients can get information about client screens using routines listed in Table 2–2. Clients can get information about images created on screens using routines listed and described in Table 2–3.

These routines are useful for supplying arguments to other routines. See the *VMS DECwindows Xlib Routines Reference Manual* for the syntax of information routines. Programming examples throughout this programming guide provide examples and descriptions of the use of information routines.

Table 2–1 Client-Server Connection Routines

Routine	Value returned
ALL PLANES	All bits set on. Used as a plane argument to a routine.
BLACK PIXEL	Pixel value that yields black on the specified screen.

(continued on next page)

Table 2–1 (Cont.) Client-Server Connection Routines

Routine	Value returned
CONNECTION NUMBER	Connection number of the specified display.
DEFAULT COLORMAP	Identifier of the default color map for allocation on the specified screen.
DEFAULT DEPTH	Depth in planes of the default root window for the specified screen.
DEFAULT GC	Default graphics context for the root window of the specified screen.
DEFAULT ROOT WINDOW	Default root window for the specified screen.
DEFAULT SCREEN	Default screen referred to by the OPEN DISPLAY routine.
DEFAULT VISUAL	Default visual data structure for the specified screen.
DISPLAY CELLS	Number of color map entries on the specified screen.
DISPLAY PLANES	Number of planes on the specified screen.
DISPLAY STRING	String passed when the display was opened. The string takes the form 0::NAME.
IMAGE BYTE ORDER	Byte order for images for each scanline unit in XY format (bitmap) or for each pixel value in Z format. If byte order is least most significant bit first, the server returns the constant LSBFirst. If the byte order is most significant bit first, the server returns the constant MSBFirst.
PROTOCOL REVISION	Minor protocol revision number that the server is using.
PROTOCOL VERSION	Version number of the protocol associated with the display.
Q LENGTH	Length of the event queue for the display. There may be events that the server has not put on the queue.
ROOT WINDOW	Identifier of the root window.
SCREEN COUNT	Number of available screens.
SERVER VENDOR	Identifier of the owner of the server implementation.
VENDOR RELEASE	Release number of the server, which is assigned by the vendor.
WHITE PIXEL	Pixel value that yields white on the specified screen.

Table 2–2 Screen Routines

Routine	Value Returned
BLACK PIXEL OF SCREEN	Black pixel value of the specified screen.
CELLS OF SCREEN	Number of color map entries for the specified screen.
DEFAULT COLORMAP OF SCREEN	Identifier of the default color map of the specified screen.
DEFAULT DEPTH OF SCREEN	Depth in planes of the specified screen.
DEFAULT GC OF SCREEN	Default graphics context of the specified screen.
DEFAULT SCREEN OF DISPLAY	Default screen of display.
DEFAULT VISUAL OF DISPLAY	Default visual type of display.
DOES BACKING STORE	Backing store is not supported in this release.
DOES SAVE UNDERS	Either true or false. True indicates the server saves the contents of windows that the client window obscures.
DISPLAY OF SCREEN	Display of the screen.
EVENT MASK OF SCREEN	Root event mask of the screen.
HEIGHT OF SCREEN	Height of screen in pixels.
HEIGHT MM OF SCREEN	Height of screen in millimeters.
MAX CMAPS OF SCREEN	Maximum number of color maps supported by the screen.
MIN CMAPS OF SCREEN	Minimum number of color maps supported by the screen.
PLANES OF SCREEN	Number of planes on the screen.
ROOT WINDOW OF SCREEN	Root window on the screen.
SCREEN OF DISPLAY	Identifier of the specified screen.
WHITE PIXEL OF SCREEN	White pixel value of the specified screen.
WIDTH OF SCREEN	Width of the screen in pixels.
WIDTH MM OF SCREEN	Width of the screen in millimeters.

Table 2–3 Image Format Routines

Routine	Value Returned
BITMAP BIT ORDER	The leftmost bit in a bitmap can be either the least or most significant bit. This routine returns either the constant LSBFirst or the constant MSBFirst.
BITMAP PAD	Number of bits by which scanlines are padded.
BITMAP UNIT	Size in bits of a bitmap unit.
DISPLAY HEIGHT	Height of the screen in pixels.

(continued on next page)

Table 2–3 (Cont.) Image Format Routines

Routine	Value Returned
DISPLAY HEIGHT MM	Height of the screen in millimeters.
DISPLAY WIDTH	Width of the display in pixels.
DISPLAY WIDTH MM	Width of the display in millimeters.

2.5 Managing Requests to the Server

Instead of sending each request to the server as the client specifies the request, Xlib buffers requests and sends them as a block to increase the efficiency of client-to-server communication. The routines listed in Table 2–4 control how requests are output from the buffer.

Table 2–4 Output Buffer Routines

Routine	Description
FLUSH	Flushes the buffer.
SET AFTER FUNCTION	Specifies the function the client calls after processing each protocol request.
SYNC	Flushes the buffer and waits until the server has received and processed all events, including errors. Use SYNC to isolate one call when debugging.
SYNCHRONIZE	Causes the server to process requests in the buffer synchronously. SYNCHRONIZE causes Xlib to generate a return after each Xlib routine completes. Use it to debug an entire client or block.

Most clients do not need to call the FLUSH routine because the output buffer is automatically flushed by calls to event management routines. Refer to Chapter 9 for more information about event handling.

3 Working with Windows

Windows receive information from users; they display graphics, text, and messages. Xlib enables a client to create multiple windows and define window size, location, and visual appearance on one or more screens.

Conflicts between clients about displaying windows are handled by a window manager, which controls the size and placement of windows and, in some cases, window characteristics such as title bars and borders. The window manager also keeps clients informed about what it is doing with their windows. For example, the window manager might tell a client that one of its windows has been resized so that the client can reformat information displayed in the window.

This chapter describes the following topics related to windows and the window manager:

- Window fundamentals—A discussion of window type, hierarchy, position, and visibility
- Creating and destroying windows—How to create and destroy windows
- Working with the window manager—How to work with the window manager to define user information concerning window management
- Mapping and unmapping windows—How to make windows visible on the screen
- Changing window characteristics—How to change the size, position, stacking order, and attributes of windows
- Getting information about windows—How to get information about window hierarchies, attributes, and geometry

3.1 Window Fundamentals

A window is an area of the screen that either receives input or receives input and displays graphics.

One type of window only receives input. Because an input-only window does not display text or graphics, it is not visible on the screen. Clients can use input-only windows to control cursors, manage input, and define regions in which the pointer is used exclusively by one client.

A second type of window both receives input and displays text and graphics.

Clients can make input-output windows visible on the screen. To make a window visible, a client first creates the window and then maps it. Mapping a window allows it to become visible on the screen. When more than one window is mapped, the windows may overlap. Window hierarchy and position on the screen determine whether or not one window hides the contents of another window.

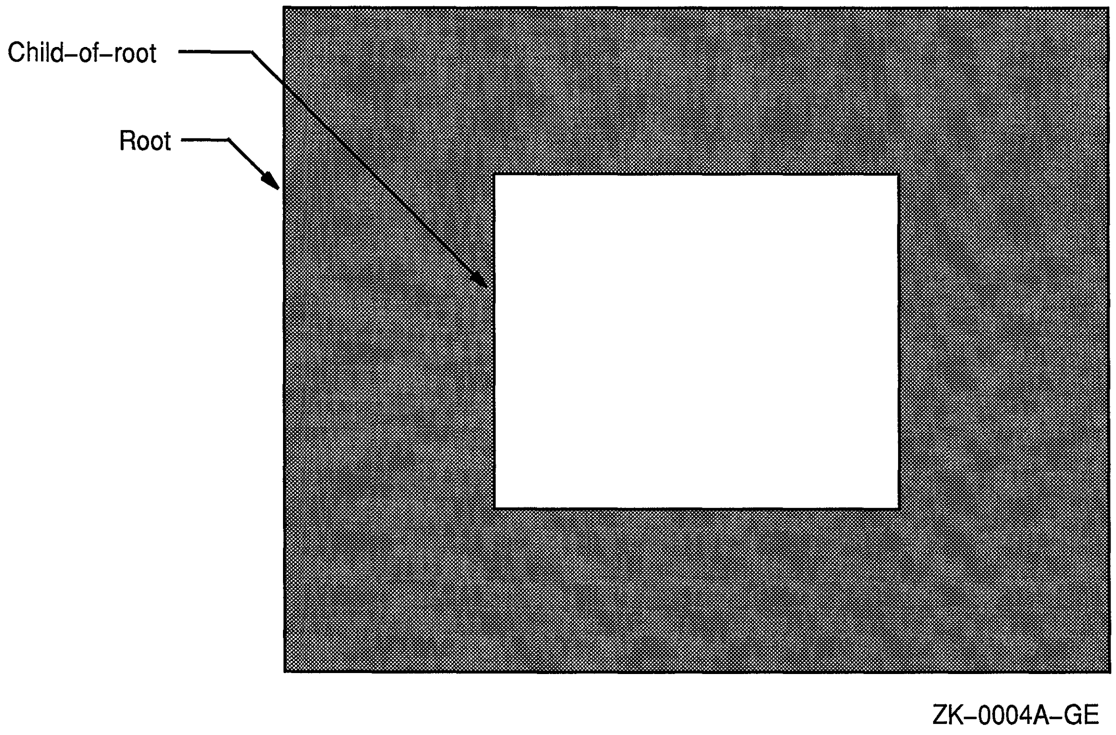
3.1.1 Window Hierarchy

Windows that clients create are part of a window hierarchy. The hierarchy determines how windows are seen. At the base of the hierarchy is the root window, which covers the entire screen when the client opens a display. All windows created after opening a display are subwindows of the root window.

When a client creates one or more subwindows of the root window, the root window becomes a **parent**. Children of the root window become parents when clients create subwindows of the children.

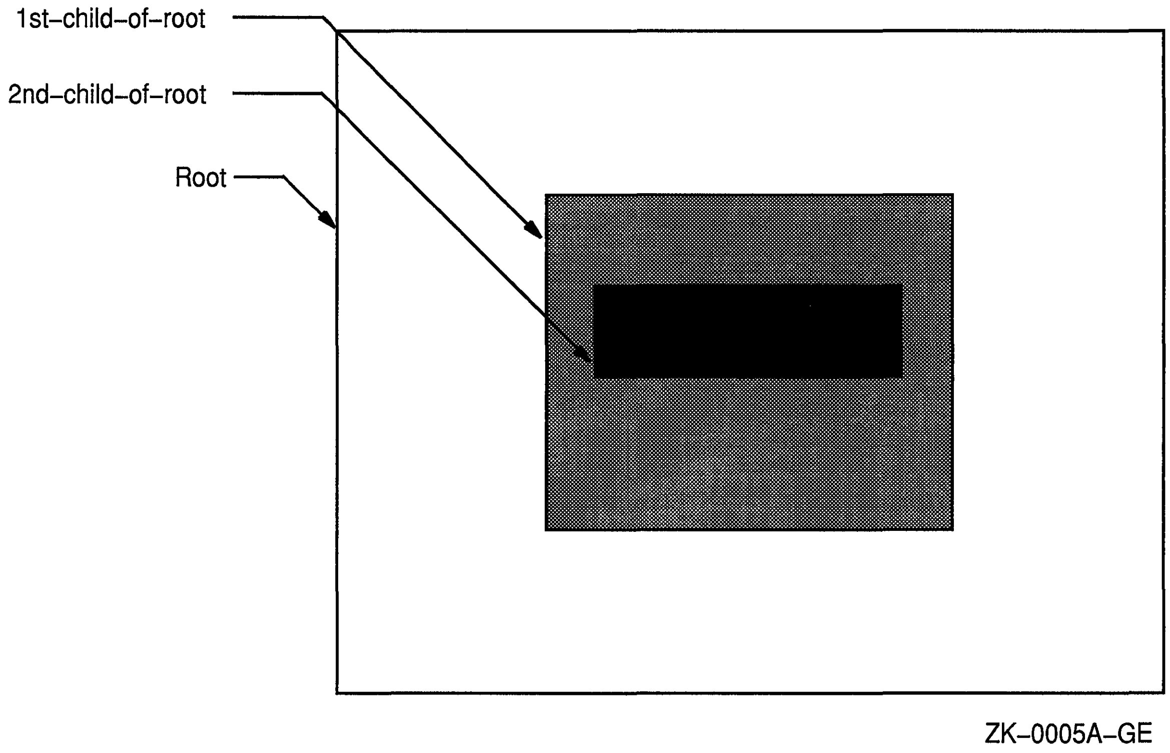
The hierarchy is structured like a stack of papers. At the bottom of the stack is the root window. Windows that clients create after opening a display are stacked on top of the root window, overlapping parts of it. For example, the window named *child-of-root* overlaps parts of the root window in Figure 3–1. The child-of-root window always touches the root window. Xlib always stacks children on top of the parents.

Figure 3–1 Root Window and One Child



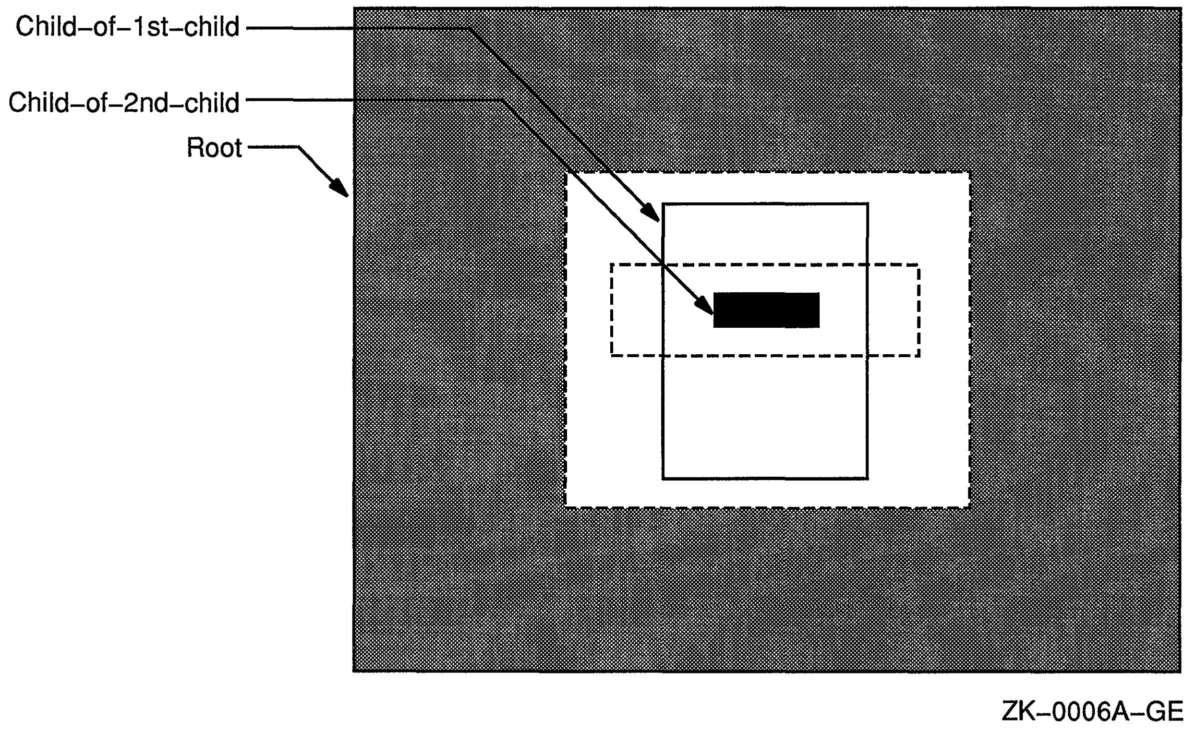
If a window has more than one child and if their borders intersect, Xlib stacks siblings in the order the client creates them, with the last sibling on top. For example, the second-level window named *2nd-child-of-root*, which was created last, overlaps the second-level window named *1st-child-of-root* in Figure 3–2.

Figure 3–2 Relationship Between Second-Level Windows



Third-level windows maintain the hierarchical relationships of their parents. The *child-of-1st-child* window overlaps *child-of-2nd-child* in Figure 3–3.

Figure 3–3 Relationship Between Third-Level Windows



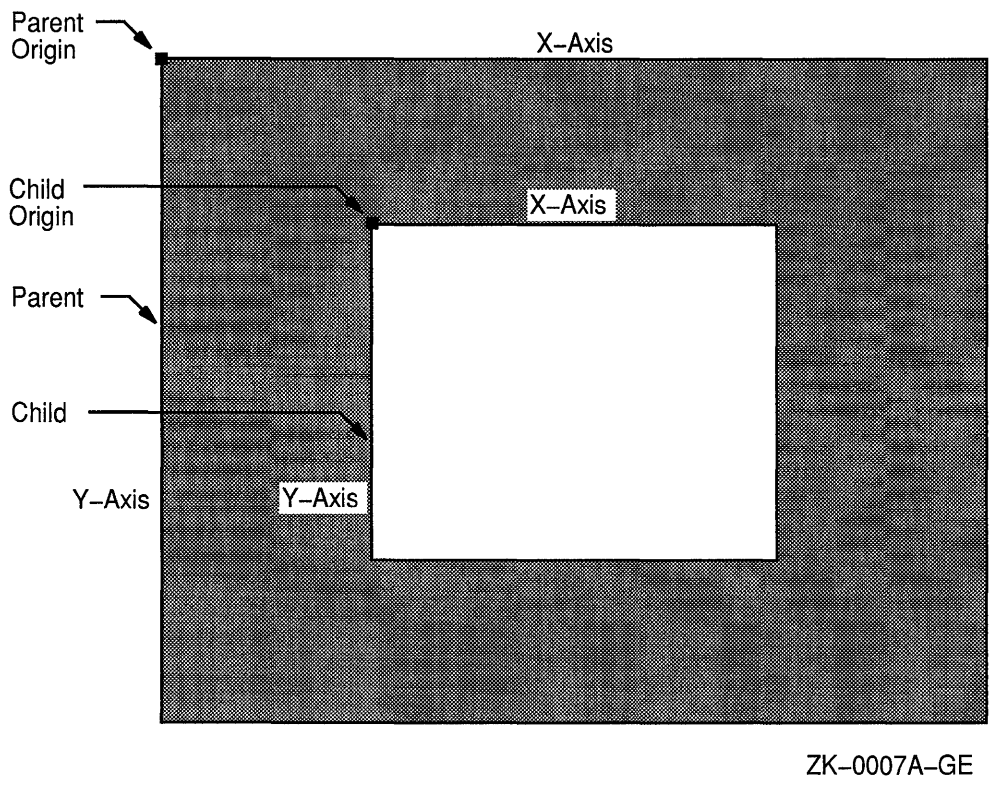
Windows created before a specified window and hierarchically related to it are ancestors of that window. For example, the root window and the window named *1st-child-of-root* are ancestors of *child-of-1st-child-of-root.*

3.1.2 Window Position

Xlib coordinates define window position on a screen and place graphics within windows. Coordinates that specify the position of a window are relative to the **origin**, the upper left corner of the parent window. Coordinates that specify the position of a graphic object within a window are relative to the origin of the window in which the graphic object is displayed.

Xlib measures length along the x axis from the origin to the right; it measures length along the y axis from the origin down. Xlib specifies coordinates in units of **pixels**, the smallest unit the server can display on a screen. Figure 3–4 illustrates the Xlib coordinate system.

Figure 3–4 Coordinate System



For more information about positioning windows, see Section 3.2. For more information about positioning graphics, see Chapter 6.

3.1.3 Window Visibility and Occlusion

A window is **visible** if one can see it on the screen. To be visible, a window must be an input-output window, it must be mapped, its ancestors must be mapped, and it must not be totally hidden by another window. When a window and its ancestors are mapped, the window is considered **viewable**. A viewable window that is totally hidden by another window is not visible.

Even though input-only windows are never visible, they can overlap other windows. An input-only window that overlaps another window is considered to **occlude** that window. Specifically, window A occludes window B if both are mapped, if A is higher in the stacking order than B, and if the rectangle defined by the outside edges of A intersects the rectangle defined by the outside edges of B.

A viewable input-output window that overlaps another window is considered to **obscure** that window. Specifically, window A obscures window B if A is a viewable input-output window, if A is higher in the stacking order than B, and if the rectangle defined by the outside edges of A intersects the rectangle defined by the outside edges of B.

3.2 Creating Windows

After opening a display, clients can create windows. As noted in the description of window fundamentals (Section 3.1), creating a window does not make it visible on a screen. To be visible, the window must meet the conditions described in Section 3.1.3.

Clients can either create windows that inherit most characteristics not relating to size or shape from their parents or define all characteristics when creating windows.

3.2.1 Using Attributes of the Parent Window

An **attribute** is a characteristic of a window not relating to size or shape, such as the window background color. The CREATE SIMPLE WINDOW routine creates an input-output subwindow that inherits the following attributes from its parent:

- Method of moving the contents of a window when the parent is moved or resized
- Instructions for saving window contents when the window obscures or is obscured by another window
- Instructions to the server regarding information that ancestors should know when a window change occurs
- Instructions to the window manager concerning map requests
- Color
- Cursor

For more information about these attributes, see Section 3.2.2.

If the parent is a root window, the new window created with the CREATE SIMPLE WINDOW routine has the following attributes:

- The server discards window contents if the window is reconfigured.
- The server discards the contents of obscured portions of the window.
- The server discards the contents of any window that the new window obscures.
- No events are specified as being of interest to the window ancestors.
- No restrictions are placed on the window manager.
- The color is identical to the parent color.
- No cursor is specified.

In addition to creating a window with attributes inherited from the parent window, the CREATE SIMPLE WINDOW routine enables clients to define the border and background attributes of the window and its position and size.

Example 3–1 illustrates creating a simple window. To make the window visible, the example includes mapping and event handling functions, which are described in Section 3.4 and Chapter 9.

Example 3–1 Creating a Simple Window

```
Window win1;
   .
   .
   .
static void doCreateWindows( )
{
❶   int win1W = 600;
    int win1H = 600;
❷   int win1X = (XWidthOfScreen(screen)-window1W)>>1;
    int win1Y = (XHeightOfScreen(screen)-window1H)>>1;

    /* Create the window */

❸   win1 = XCreateSimpleWindow(dpy, XRootWindowOfScreen(screen),
         win1X, win1Y, win1W, win1H, 10, XBlackPixelOfScreen(screen),
         XWhitePixelOfScreen(screen);
}
```

❶ Assign window width and height the value of 600 (pixels) each.

❷ The client specifies the position of the window using two display information routines, WIDTH OF SCREEN and HEIGHT OF SCREEN. The **x** and **y** coordinates define the top left outside corner of the window borders relative to the inside of the parent border. In this case, the parent is the root window, which does not have a border.

❸ The CREATE SIMPLE WINDOW routine call has the following format:

```
window_id = XCreateSimpleWindow(display, parent_id,
        x_coord, y_coord, width, height, border_width,
        border_id, background_id)
```

The client specifies a black border ten pixels wide, a white background, and a size of 600 by 600 pixels.

The window manager overrides border width and color.

CREATE SIMPLE WINDOW returns a unique identifier, *win1*, used in subsequent calls related to the window.

3.2.2 Defining Window Attributes

To create a window whose attributes are different from the parent window, use the CREATE WINDOW routine. The CREATE WINDOW routine enables clients to specify the following window attributes when creating an input-output window:

- Default contents of an input-output window
- Border of an input-output window
- Treatment of the window when it or its relative is obscured
- Treatment of the window when it or its relative is moved

- Information the window receives about operations associated with other windows
- Color
- Cursor

Clients creating input-only windows can define the following attributes:

- Treatment of the window when it or its relative is moved
- Information the window receives about operations associated with other windows
- Cursor

Specifying other attributes for an input-only window causes the server to generate an error. Input-only windows cannot have input-output windows as children.

Use the following method to define window attributes:

1. Assign values to the relevant members of a set window attributes data structure.
2. Indicate the defined attribute by specifying the appropriate flag in the **value_mask** argument of the CREATE WINDOW routine. If more than one attribute is to be defined, indicate the attributes by doing a bitwise OR on the appropriate flags.

The following illustrates the set window attributes data structure:

```
typedef struct {
    Pixmap background_pixmap;
    unsigned long background_pixel;
    Pixmap border_pixmap;
    unsigned long border_pixel;
    int bit_gravity;
    int win_gravity;
    int backing_store;
    unsigned long backing_planes;
    unsigned long backing_pixel;
    Bool save_under;
    long event_mask;
    long do_not_propagate_mask;
    Bool override_redirect;
    Colormap colormap;
    Cursor cursor;
} XSetWindowAttributes;
```

Table 3–1 describes the members of the data structure.

Table 3–1 Set Window Attributes Data Structure Members

Member Name	Contents
background_pixmap	Defines the window background. The background_pixmap member can assume one of three possible values: pixmap identifier, the constant None (default), or the constant ParentRelative. If the client specifies a pixmap identifier, a pixmap defines the window background. The pixmap must have the same root and number of bits per pixel as the window, but can be any size. For more information about creating pixmaps, see Chapter 7. If the client specifies the constant None (default), the window has no defined background. If the parent has no defined background, neither does the window being created. If the client specifies the constant ParentRelative, the background of the window is identical to the background of its parent. In this case, the window must have the same number of bits per pixel as the parent. If the background value of the window is ParentRelative and the parent background is None, the window being created has no defined background. The server does not copy the parent background; instead, it reexamines the parent background each time the client needs the window background. For a background that is identical to the parent background, the origin of the pixmap used to paint the background, the background tile, always aligns with the origin of the parent background tile origin. Otherwise, the background tile origin is always the window origin. If the client alters the pixmap after using it for the background, the results are unpredictable because the server might either make a copy of the pixmap used to draw the background, or it might refer to the pixmap directly. Free the background pixmap when the client no longer needs to refer to it. In particular, free the pixmap after setting it into the window but before destroying the window. When regions of the window are exposed and the server has not retained their contents, the server automatically tiles the regions with the background pixmap if the client specified a pixmap identifier or the constant ParentRelative. If the client specified the constant None, the server leaves the previous screen contents in place, provided the window and its parent have the same number of bits per pixel. Otherwise, the initial contents of the exposed region are undefined.
background_pixel	Specifying a value for the background_pixel member causes the server to override the background_pixmap member. This is equivalent to specifying a pixmap of any size filled with the background pixel and used to paint the window background.

(continued on next page)

Table 3–1 (Cont.) Set Window Attributes Data Structure Members

Member Name	Contents
border_pixmap	Defines the window border. The following conditions apply: • The border tile origin is always the same as the background tile origin. • The border pixmap and the window must have the same root and the same number of bits per pixel. Otherwise, the server issues an error. • Clients can specify a pixmap of any size. Using some sizes, however, increases performance. • The default copies the border pixmap from the parent. If the client specifies the constant CopyFromParent, the parent border pixmap is copied. The window must have the same number of bits per pixel as the parent, or the server issues an error. Subsequent changes to the parent do not affect the child. If the client alters the pixmap after using it for the border, the results are unpredictable because the server may either make a copy of the pixmap used to draw the border, or it may refer to the pixmap directly. Because output to a window is always limited or clipped to the inside of the window, graphics operations are never affected by the window border.
border_pixel	Specifying a value for border_pixel causes the server to override the border_pixmap member. This is equivalent to specifying a pixmap of any size filled with the border pixel and used to paint the window border.
bit_gravity	Defines how the contents of the window should be moved when the window is resized. By default, the server does not retain window contents. For more information about bit gravity, see Section 3.6.
win_gravity	Defines how the server should reposition the newly created window when its parent window is resized. By default, the server does not move the newly created window. For more information about window gravity, see Section 3.6.
backing_store	Provides a hint to the server about how the client wants it to manage obscured portions of the window. In this release, clients must maintain window contents.
backing_planes	Indicates (with bits set to one) which bit planes of the window hold dynamic data that must be preserved if the window obscures or is obscured by another window. In this release, clients must maintain data to be preserved.
backing_pixel	Defines what values to use in planes not specified by the backing_planes member. The server is free to save only specified bit planes and to regenerate the remaining planes with the specified pixel value. Bits that extend beyond the number per pixel of the window are ignored. In this release, clients must maintain values.
save_under	Setting the save_under member to true informs the server that the client would like the contents of the screen saved when the window obscures them. Saving the contents of obscured portions of the screen is not guaranteed.

(continued on next page)

Table 3–1 (Cont.) Set Window Attributes Data Structure Members

Member Name	Contents
event_mask	Defines which types of events associated with the window the server should report to the client. For more information about defining event types, see Chapter 9. Following are events about which the client can state an interest:

Event Type	Description
Button	Motion, button press and release, exclusive input
Color	Change in color map
Window	Entry into and exit from a window
Exposure	Exposure of a previously obscured window
Input focus	Change in window that receives keyboard input
Keyboard and keys	Change in keyboard state, and key press or release
Pointer	Motion
Property	Change in window characteristics
Structure	Notification and control of requests from clients

Member Name	Contents
do_not_propagate_mask	Defines which kinds of events should not be propagated to ancestors. For more information about managing events, see Chapter 9.
override_redirect	Specifies whether calls to map and configure the window should override a request by another client to redirect those calls. For more information about redirecting calls, see Chapter 9. Typically, this is used to inform a window manager not to tamper with the window, for example when the client is creating and mapping a menu.
color map	Specifies the color map, if any, that best reflects the colors of the window. The color map must have the same visual type as the window. If it does not, the server issues an error. For more information about the color map and visual types, see Chapter 5.
cursor	Specifying a value for the cursor member causes the server to use a particular cursor when the pointer is in the window.

Table 3–2 lists default values for the set window attributes data structure.

Table 3–2 Default Values of the Set Window Attributes Data Structure

Member	Default Value
background_pixmap	None
background_pixel	Undefined
border_pixmap	Copied from the parent window
border_pixel	Undefined
bit_gravity	Window contents not retained
win_gravity	Window not moved

(continued on next page)

Table 3–2 (Cont.) Default Values of the Set Window Attributes Data Structure

Member	Default Value
backing_store	Window contents not retained
backing_planes	All 1s
backing_pixel	0
save_under	False
event_mask	Empty set
do_not_propagate_mask	Empty set
override_redirect	False
colormap	Copied from parent
cursor	None

Xlib assigns a flag for each member of the set window attributes data structure to facilitate referring to the members, as listed in Table 3–3.

Table 3–3 Set Window Attributes Data Structure Flags

Flag Name	Set Window Attributes Member
CWBackPixmap	background_pixmap
CWBackPixel	background_pixel
CWBorderPixmap	border_pixmap
CWBorderPixel	border_pixel
CWBitGravity	bit_gravity
CWWinGravity	win_gravity
CWBackingStore	backing_store
CWBackingPlanes	backing_planes
CWBackingPixel	backing_pixel
CWOverrideRedirect	override_redirect
CWSaveUnder	save_under
CWEventMask	event_mask
CWDontPropagate	do_not_propagate
CWColormap	colormap
CWCursor	cursor

Example 3–2 illustrates how clients can define window attributes while creating input-output windows with the CREATE WINDOW routine. The program creates a parent window and two children windows. The hierarchy of the subwindows is determined by the order in which the program creates them. In this case, *subwin1* is superior to *subwin2*, which is created last.

Example 3–2 Defining Attributes When Creating Windows

```
Window window, subwindow1,subwindow2;
int n;
    .
    .
    .
static void doCreateWindows( )
{
    int windowW = 600;
    int windowH = 600;
    int windowX = (WidthOfScreen(screen)-windowW)>>1;
    int windowY = (HeightOfScreen(screen)-windowH)>>1;
    int subwindow1X = 150;
    int subwindow1Y = 100;
    int subwindow1W = 300;
    int subwindow1H = 400;
    int subwindow2X = 275;
    int subwindow2Y = 125;
    int subwindow2W =  50;
    int subwindow2H = 150;
❶   XSetWindowAttributes xswa;
    .
    .
    .
    /* Create the window window */

❷   xswa.event_mask = ExposureMask | ButtonPressMask;
    xswa.background_pixel = doDefineColor(1);

❸   window = XCreateWindow(dpy, XRootWindowOfScreen(screen),
        windowX, windowY, windowW, windowH, 0,
        XDefaultDepthOfScreen(screen), InputOutput,
        XDefaultVisualOfScreen(screen), CWEventMask | CWBackPixel, &xswa);

    /* Create the window subwindow1 */

    xswa.background_pixel = doDefineColor(3);

    subwindow1 = XCreateWindow(dpy, window, subwindow1X, subwindow1Y, subwindow1W,
        subwindow1H, 4, XDefaultDepthOfScreen(screen), InputOutput,
        XDefaultVisualOfScreen(screen), CWEventMask | CWBackPixel, &xswa);

    /* Create the window subwindow2 */

    xswa.background_pixel = doDefineColor(3);

    subwindow2 = XCreateWindow(dpy, window, subwindow2X, subwindow2Y, subwindow2W,
        subwindow2H, 4, XDefaultDepthOfScreen(screen), InputOutput,
        XDefaultVisualOfScreen(screen), CWEventMask | CWBackPixel, &xswa);
}
    .
    .
    .
static int doDefineColor(n)
{
    .
    .
    .
```

❶ Allocate storage for a set window attributes data structure used to define window attributes.

❷ Set the attributes of the parent window. The client indicates an interest in window exposure and button press events. For more information about events, see Chapter 9.

The client defines the window background by calling the client-defined *doDefineColor* routine. For more information about defining colors, see Chapter 5.

❸ The CREATE WINDOW routine call has the following format:

```
window_id_return=XCreateWindow(display, parent_id,
          x_coord, y_coord, width, height, border_width,
          depth, class, visual_struc, attributes_mask,
          attributes)
```

The depth of a window is its number of bits per pixel. The call passes a display information routine to indicate that the client wants the parent window depth to be identical to the display depth.

The window class can be either input only or input-output, specified by the following constants:

- InputOnly
- InputOutput

If the window is the same class as the parent, pass the constant **CopyFromParent**.

The visual type indicates how the window displays color values. For more information about visual types, see Chapter 5.

3.3 Destroying Windows

When a client no longer needs a window, the client should destroy it using either the DESTROY WINDOW or the DESTROY SUBWINDOWS routine. DESTROY WINDOW destroys a specified window and all its subwindows. DESTROY SUBWINDOWS destroys all subwindows of a specified window in bottom to top stacking order.

Destroying a window frees all storage allocated for that window. If the window is mapped to the screen, the server notifies applications using the window that it has been destroyed.

3.4 Mapping and Unmapping Windows

After creating a window, the client can map it to a screen using the MAP WINDOW or MAP SUBWINDOWS routine. Mapping generally makes a window visible at the location the client specified when creating it. Part or all of the window is not visible when the following conditions occur:

- One or more windows higher in the stacking order obscures it
- One or more window ancestors is not mapped
- The new window extends beyond the boundary of its parent

MAP WINDOW maps a window. If the window is an inferior, and one or more of its ancestors has not been mapped, the server considers the window to be mapped after the call, even though the window is not visible on the screen. The window becomes visible when its ancestors are mapped.

To map all subwindows of a specified window in top to bottom order, use MAP SUBWINDOWS. Using the MAP SUBWINDOWS routine to map several windows may be more efficient than calling the MAP WINDOW routine to map each window. The MAP SUBWINDOWS routine enables the server to map all of the windows at one time instead of mapping a single window with the MAP WINDOW routine.

To ensure that the window is completely visible, use the MAP RAISED routine. MAP RAISED reorders the stack with the window on top and then maps the window. Example 3–3 illustrates how a window is mapped and raised to the top of the stack.

Example 3–3 Mapping and Raising Windows

```
Window window,subwindow1,subwindow2;

    /* Create windows in the following order: window, subwindow2, subwindow1 */
    .
    .
    .
static void doMapWindows( )
{
    XMapWindow(dpy, window);
❶   XMapWindow(dpy, subwindow2);
❷   XMapRaised(dpy, subwindow1);
}
```

❶ In this example, the client created *subwindow1* after *subwindow2*, putting *subwindow1* at the top of the stack.

Consequently, whether *subwindow2* were to be mapped before or after *subwindow1*, *subwindow1* would obscure *subwindow2*.

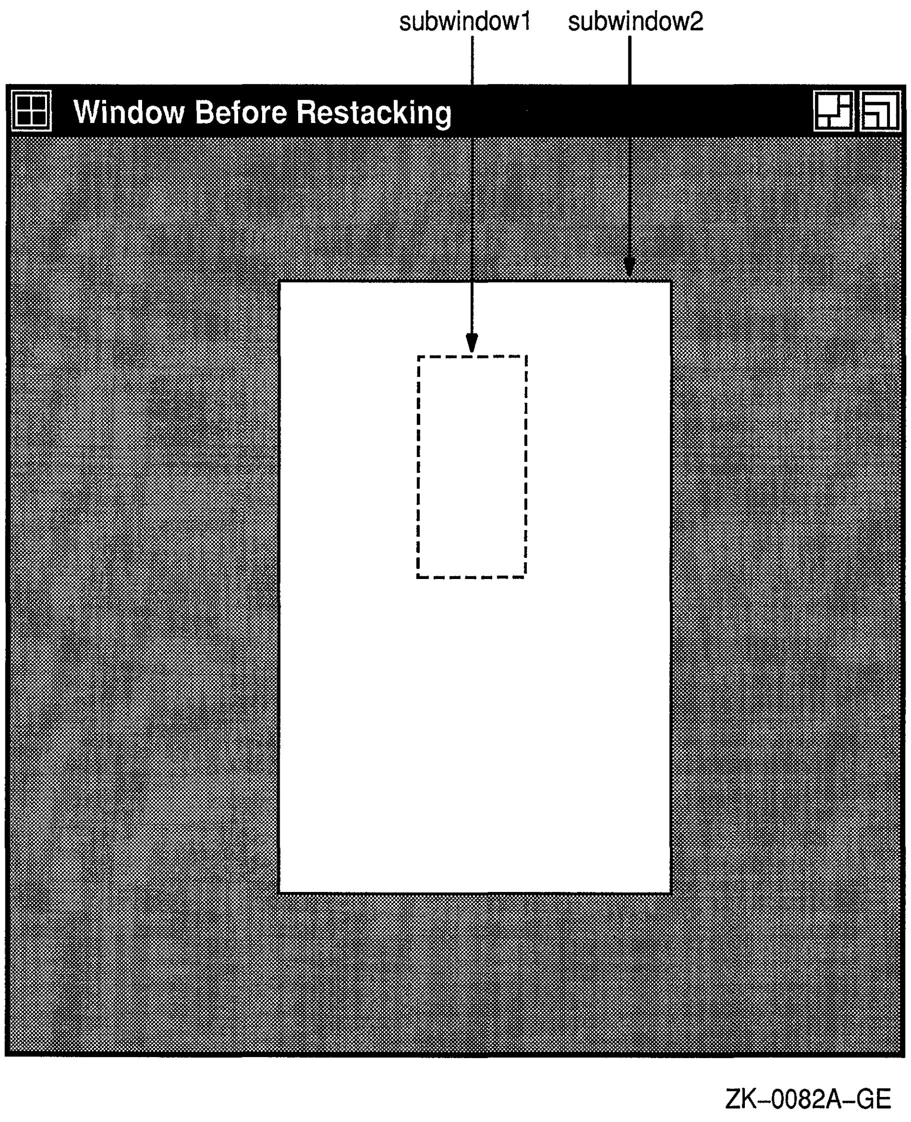
The effect is illustrated in Figure 3–5.

❷ Mapping and raising *subwindow2* moves it to the top of the stack. It is now visible, as Figure 3–6 illustrates.

When the client no longer needs a window mapped to the screen, call UNMAP WINDOW. If the window is a parent, its children are no longer visible after the call, although they are still mapped. The children become visible when the parent is mapped again.

To unmap all subwindows of a specified window, use UNMAP SUBWINDOWS. UNMAP SUBWINDOWS results in an UNMAP WINDOW call on all subwindows of the parent, from bottom to top stacking order.

Figure 3–5 Window Before Restacking

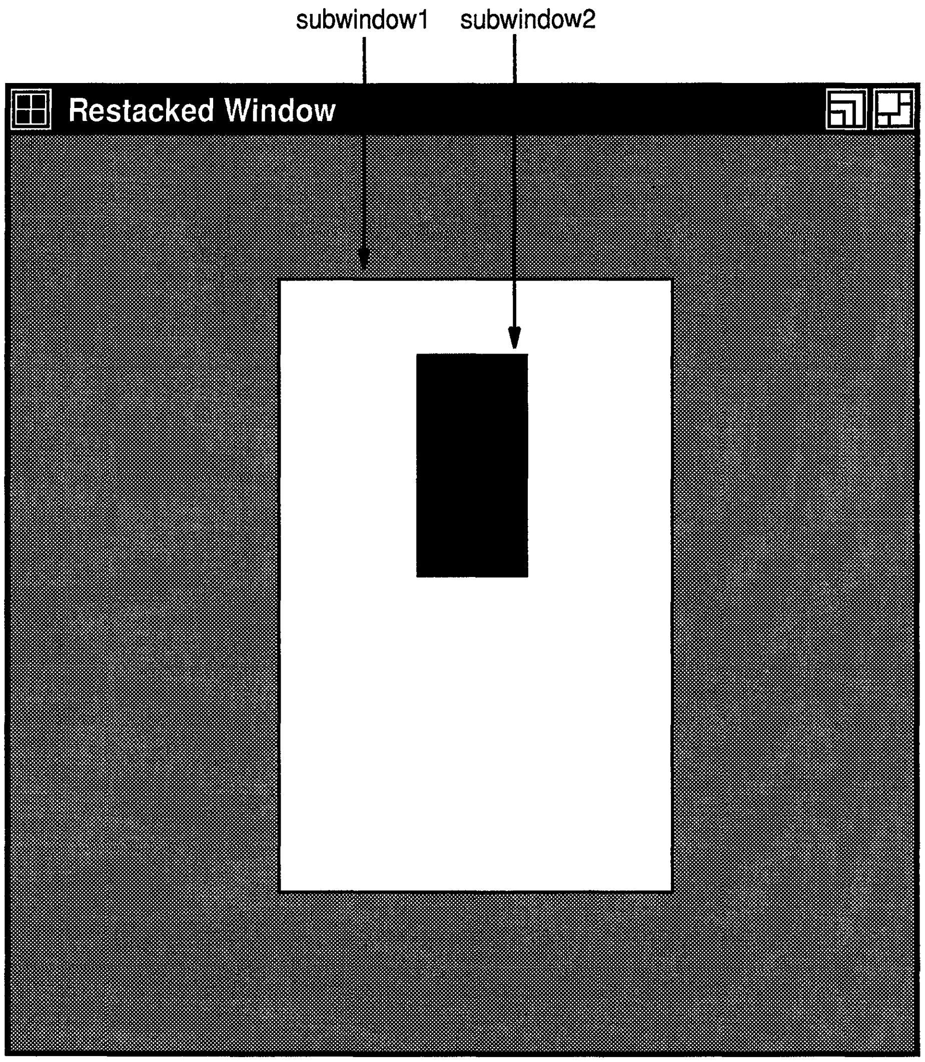


3.5 Associating Properties with Windows

Xlib enables clients to associate data with a window. This data is considered a **property** of the window. For example, a client could store text as a window property. Although a property must be data of only one type, it can be stored in 8-bit, 16-bit, and 32-bit formats.

Xlib uses **atoms** to uniquely identify properties. An atom is a string paired with an identifier. For example, a client could use the atom XA_WM_ICON_NAME to name a window icon stored for later use. The atom XA_WM_ICON_NAME pairs the string XA_WM_ICON_NAME with a value, 37, that uniquely identifies a property.

Figure 3–6 Restacked Window



ZK–0080A–GE

In DECW$INCLUDE:XATOMS.H, VMS DECwindows includes predefined atoms such as XA_WM_ICON_NAME for commonly used properties. Table 3–4 lists by function all predefined atoms except those used to identify font properties and atoms used to communicate with the window manager. See Table 3–6 for a list of atoms related to window management. See Chapter 8 for a list of atoms related to fonts.

Table 3–4 Predefined Atoms

For Global Selection	
XA_PRIMARY	XA_SECONDARY
For Cut Buffers	
XA_CUT_BUFFER0	XA_CUT_BUFFER1
XA_CUT_BUFFER2	XA_CUT_BUFFER3
XA_CUT_BUFFER4	XA_CUT_BUFFER5
XA_CUT_BUFFER6	XA_CUT_BUFFER7
For Color Maps	
XA_RGB_COLOR_MAP	XA_RGB_BEST_MAP
XA_RGB_BLUE_MAP	XA_RGB_RED_MAP
XA_RGB_GREEN_MAP	XA_RGB_GRAY_MAP
XA_RGB_DEFAULT_MAP	
For Resources	
XA_RESOURCE_MANAGER	XA_ARC
XA_ATOM	XA_BITMAP
XA_CARDINAL	XA_COLORMAP
XA_CURSOR	XA_DRAWABLE
XA_FONT	XA_INTEGER
XA_PIXMAP	XA_POINT
XA_RECTANGLE	XA_STRING
XA_VISUALID	XA_WINDOW

In addition to providing predefined atoms, Xlib enables clients to create atom names of their own. To create an atom name, use the INTERN ATOM routine, as in the following example:

```
    .
    .
    .
    Atom atom_id;
    char *name = "MY_ATOM";
    Bool if_exists;

    atom_id =  XInternAtom(dpy, name, if_exists);
    .
    .
    .
```

The routine returns an identifier associated with the string MY_ATOM. Xlib also returns the value of false to *if_exists* if the atom does not exist in the atom table.

To get the name of an atom, use the GET ATOM NAME routine, as in the following example:

```
    .
    .
    .
    char name;
    Atom atom_id = 39;

    name = XGetAtomName(dpy, atom_id);
    .
    .
    .
```

The routine returns a string associated with the atom identifier.

Xlib enables clients to change, obtain, update, and interchange properties. Example 3–4 illustrates exchanging properties between two subwindows. The example uses the CHANGE PROPERTY routine to set a property on the parent window and the GET PROPERTY routine to get the data from the parent window.

Example 3–4 Exchanging Window Properties

```
#define windowWidth      600
#define windowHeight     600
#define subwindowWidth   300
#define subwindowHeight  150
#define true 1
    .
    .
    .

static void doCreateWindows( )
{
    int winW = windowWidth;
    int winH = windowHeight;
    int winX = 100;
    int winY = 100;
    int subwindow1X = 150;
    int subwindow1Y = 100;
    int subwindow2X = 150;
    int subwindow2Y = 350;
    XSetWindowAttributes xswa;

    /* Create the win window */

    xswa.event_mask = ExposureMask | ButtonPressMask | PropertyChangeMask;
    xswa.background_pixel = doDefineColor(1);

    win = XCreateWindow(dpy, RootWindowOfScreen(screen),
        winX, winY, winW, winH, 0,
        DefaultDepthOfScreen(screen), InputOutput,
        DefaultVisualOfScreen(screen), CWEventMask | CWBackPixel, &xswa);

    /* Create the subwindows */
    xswa.event_mask = ExposureMask| ButtonPressMask;
    xswa.background_pixel = doDefineColor(2);
```

(continued on next page)

Example 3–4 (Cont.) Exchanging Window Properties

```
    subwin1 = XCreateWindow(dpy, win, subwindow1X, subwindow1Y, subwindowWidth,
        subwindowHeight, 0, DefaultDepthOfScreen(screen), InputOutput,
        DefaultVisualOfScreen(screen), CWEventMask | CWBackPixel, &xswa);
    subwin2 = XCreateWindow(dpy, win, subwindow2X, subwindow2Y, subwindowWidth,
        subwindowHeight, 0, DefaultDepthOfScreen(screen), InputOutput,
        DefaultVisualOfScreen(screen), CWEventMask | CWBackPixel, &xswa);
    .
    .
    .
/****************** doHandleEvents **********************/
static void doHandleEvents( )
{
    XEvent event;

    for ( ; ; ) {
        XNextEvent(dpy, &event);
        switch (event.type) {
            case Expose:                doExpose(&event); break;
            case ButtonPress:           doButtonPress(&event);break;
            case PropertyNotify:        doPropertyNotify(&event);break;
        }
    }
}
    .
    .
    .
/***** Handle button presses *******/
static void doButtonPress(eventP)
XEvent *eventP;
{
    char *property_data = "You clicked MB1";

    if (eventP->xbutton.button == Button2) sys$exit(1);
    if (eventP->xbutton.window == subwin1 && eventP->xbutton.button == Button1)
❶       XChangeProperty(dpy, win, XA_CUT_BUFFER0, XA_STRING, 16,
            PropModeReplace, property_data, 15);
    return;
}
/***** Get the property and draw text into the subwindow *******/
static void doPropertyNotify(eventP)
XEvent *eventP;
{
    long offset = 0;
    long length = 1000;
    Atom type_returned;
    int format_returned;
    unsigned long num_items_returned, bytes_remaining;
    unsigned char *property_returned;

    if (eventP->xproperty.atom == XA_CUT_BUFFER0){
❷       XGetWindowProperty(dpy, win, XA_CUT_BUFFER0, offset, length,
            true, XA_STRING, &type_returned, &format_returned,
            &num_items_returned, &bytes_remaining, &property_returned);
```

(continued on next page)

Example 3–4 (Cont.) Exchanging Window Properties

```
❸       XDrawString(dpy, subwin2, gc, 75, 75, property_returned,
            num_items_returned);
    }
    return;
}
```

❶ When the user clicks MB1 in subwindow *subwin1*, the client calls the CHANGE PROPERTY routine. CHANGE PROPERTY causes the server to change the property identified by the atom XA_CUT_BUFFER0 to the value specified by *property_data*. The property is associated with the parent window, *win*.

When changing properties, clients can specify how the server should treat the property. If the client specifies the constant **PropModeReplace**, the server discards the previous property. If the client specifies the constant **PropModePrepend**, the server inserts the new data at the beginning of the existing property data. If the client specifies the constant **PropModeAppend**, the server inserts the new data at the end of the existing property data.

Changing the property causes the server to send a property notify event to *win*. For information about event handling, see Chapter 9.

❷ After checking to ensure that the changed property is the one to obtain, the client calls the GET WINDOW PROPERTY routine.

❸ After getting the string data from the parent window, the client uses it to write text in *subwin2*. For information about writing text, see Chapter 8.

In addition to the GET WINDOW PROPERTY routine, Xlib includes the property-management routines described in Table 3–5.

Table 3–5 Routines for Managing Properties

Routine	Description
LIST PROPERTIES	Returns a list of properties defined for a specified window.
ROTATE WINDOW PROPERTIES	Rotates the properties of a specified window and generates a property notify event. For more information about property notify events, see Chapter 9.
DELETE PROPERTY	Deletes a specified property.

3.5.1 Using Properties to Communicate with the Window Manager

Xlib provides predefined atoms to enable clients to communicate hints to the window manager about the following:

- Window names
- Icon names
- Pixmaps used to define window icons
- Commands used to start the application
- Position and size of windows in their startup state
- Initial state of windows
- Input that windows accept
- Names used to retrieve application resources

Table 3–6 describes the atom names, data types, and formats of these properties.

Table 3–6 Atom Names of Window Manager Properties

Atom	Data Type	Format	Description
XA_WM_NAME	STRING	8	Application name
XA_WM_ICON_NAME	STRING	8	Icon name
XA_WM_NORMAL_HINTS	WM_SIZE_HINTS	32	Size hints for a window in its normal state
XA_WM_ZOOM_HINTS	WM_SIZE_HINTS	32	Size hints for a zoomed window
XA_WM_HINTS	WM_HINTS	32	Hints about keyboard input, initial state, icon pixmap, icon window, icon position, and icon mask
XA_WM_COMMAND	STRING	8	Command used to start the client
XA_WM_ICON_SIZE	WM_ICON_SIZE	32	Specifies the icon size supported by the window manager
XA_WM_CLASS	STRING	32	Allows window manager to obtain the application resources from the resource database
XA_WM_TRANSIENT_FOR	WINDOW	32	Indicates that a window, such as a dialog box, is transient

Xlib provides the following methods for using the properties described in Table 3–6 to communicate with the window manager:

- Defining properties with the SET WM HINTS routine—SET WM HINTS uses the WM hints data structure to define hints about keyboard input, initial state of the window, icon pixmap, icon window, icon position, icon mask, and window group.

- Using convenience routines to communicate with the window manager—Xlib includes routines that enable clients to communicate individual hints about window names, window icon names, and window classes.
- Providing and obtaining hints about the size and position of windows—Xlib routines communicate information about the size and position of windows.
- Changing the values of a property—Xlib includes a routine to change the value of an existing property.

Note that it is not guaranteed that the window manager will apply window manager hints.

This section describes how to use properties to communicate with the window manager.

3.5.1.1 Defining Properties Using the SET WM HINTS Routine

Use the SET WM HINTS routine to provide the window manager with hints about keyboard input, initial window state, icon pixmap, icon window, icon position, icon mask, and window group. A window manager can use the window group property to treat a set of windows as a group. For example, if a client manipulates multiple children of the root window, SET WM HINTS enables the client to provide enough information so that a window manager can make all windows into icons, rather than just one window.

Xlib provides a WM hints data structure to enable clients to specify these hints easily. The following illustrates the WM hints data structure:

```
typedef struct {
        long flags;
        Bool input;
        int initial_state;
        Pixmap icon_pixmap;
        Window icon_window;
        int icon_x, icon_y;
        Pixmap icon_mask;
        XID window_group;
} XWMHints;
```

Table 3–7 defines the members of the data structure.

Table 3–7 WM Hints Data Structure Members

Member Name	Contents
flags	Specifies the members of the data structure that are defined.
input	Indicates whether or not the client relies on the window manager to get keyboard input.
initial_state	Defines how the window should appear in its initial configuration. Possible initial states are as follows:

Constant Name	Description
DontCareState	Client is not interested in the initial state
NormalState	Initial state used most often
ZoomState	Window starts zoomed
IconicState	Window starts as an icon
InactiveState	Window is seldom used

Member Name	Contents
icon_pixmap	Identifies the pixmap used to create the window icon.
icon_window	Specifies the window to be used as an icon.
icon_x	Specifies the initial x-coordinate of the icon position.
icon_y	Specifies the initial y-coordinate of the icon position.
icon_mask	Specifies the pixels of the icon pixmap used to create the icon.
window_group	Specifies that a window belongs to a group of other windows.

3.5.1.2 Defining Individual Properties

Xlib includes routines to enable clients to define individual properties for communicating with the window manager about window names, icon names, and window classes.

To define a window name, use the STORE NAME routine. The sample program in Chapter 1 uses the STORE NAME routine to define the name of its parent window, as follows:

```
XStoreName(dpy, window1, "A Sample Xlib Program");
```

To get the name of a window, use the FETCH NAME routine. The routine either returns the name of the specified window or sets the value of the XA_WM_NAME property to null.

The SET ICON NAME and GET ICON NAME routines define and get the name of a window icon.

To define and get the class of a specified window, use the SET CLASS HINT and GET CLASS HINT routines. The routines refer to the class hint data structure, which has the following form:

```
typedef struct {
        char *res_name;
        char *res_class;
} XClassHint;
```

Table 3–8 defines the members of the data structure.

Table 3–8 Class Hint Data Structure Members

Member Name	Contents
res_name	Defines the name of the window. The name defined in this data structure may differ from the name defined by the XA_WM_NAME property. The XA_WM_NAME property specifies what should be displayed in the title bar. Consequently, it may contain a temporary name, as in the name of a file a client currently has in a buffer. In contrast to XA_WM_NAME, the res_name member defines the formal window name that clients should use when retrieving resources from the resource database.
res_class	Defines the class of the window.

At times, clients may need to indicate to the window manager that a top-level window is really only a transient window. For instance, a client may communicate to the window manager that the window is a dialog box mapped on behalf of another window. To communciate this, a client calls the SET TRANSIENT FOR HINT routine. The routine sets the XA_WM_TRANSIENT_FOR property of the transient window and associates the transient window with a main window. To obtain the XA_WM_TRANSIENT_FOR property for a specified window, call the GET TRANSIENT FOR HINT routine.

To define the command that invokes an application in a specified window, use the SET COMMAND routine.

3.5.1.3 Providing Size Hints

Xlib provides routines to communicate with the window manager about the size and position of windows in their normal and zoomed startup states. Use the following method to specify the size and position of a window in its usual startup state:

1 Assign values to the relevant members of the size hints data structure, including the flags member, which specifies the members of the data structure that are defined. Table 3–9 lists the flags.

2 Call the SET NORMAL HINTS routine

Table 3–9 Set Window Attributes Data Structure Flags

Flag Name	Size Hints Member
USPosition	User-specified position of the window
USSize	User-specified size of the window
PPosition	Client-specified position
PSize	Client-specified size

(continued on next page)

Table 3–9 (Cont.) Set Window Attributes Data Structure Flags

Flag Name	Size Hints Member
PMinSize	Client-specified minimum size of the window
PMaxSize	Client-specified maximum size of the window
PResizeInc	Client-specified increments for resizing the window
PAspect	Client-specified minimum and maximum aspect ratios
PAllHints	The bitwise OR of PPosition, PSize, PMinSize, PMaxSize, PResizeInc, and PAspect

The following illustrates the size hints data structure:

```
typedef struct {
        long flags;
        int x, y;
        int width, height;
        int min_width, min_height;
        int max_width, max_height;
        int width_inc, height_inc;
        struct {
                int x;
                int y;
        min_aspect, max_aspect;
}
XSizeHints;
```

Table 3–10 describes its contents.

Table 3–10 Size Hints Data Structure Members

Member Name	Contents
flags	Defines which members the client is assigning values to.
x	Specifies the x-coordinate that defines window position.
y	Specifies the y-coordinate that defines window position.
width	Defines the width of the window.
height	Defines the height of the window.
min_width	Specifies the minimum useful width of the window.
min_height	Specifies the minimum useful height of the window.
max_width	Specifies the maximum useful width of the window.
max_height	Specifies the maximum useful height of the window.
width_inc	Defines the increments by which the width of the window can be resized.
height_inc	Defines the increments by which the height of the window can be resized.
min_aspect_x	With the min_aspect y member, specifies the minimum aspect ratio of the window.

(continued on next page)

Table 3–10 (Cont.) Size Hints Data Structure Members

Member Name	Contents
min_aspect_y	With the min_aspect x member, specifies the minimum aspect ratio of the window.
max_aspect_x	With the max_aspect y member, specifies the maximum aspect ratio of the window.
max_aspect_y	With the max_aspect_x member, specifies the maximum aspect ratio of the window. Setting the minimum and maximum aspects indicates the preferred range of the size of a window. An aspect is expressed in terms of a ratio between x and y. For example, if the minimum aspect of x is 1 and y is 2, and the maximum aspect of x is 2 and y is 5, then the minimum window size is a ratio of 1/2, and the maximum is a ratio of 2/5. In this case, a window could have a width of 300 pixels and a height of 600 pixels minimally, and maximally a width of 600 pixels and a height of 1500 pixels.

The following illustrates using the size hints data structure to set the normal window manager hints for a window:

```
    .
    .
    .
static void doWMHints( )
{
    XSizeHints xsh;

    /* Define the size and name of the window window1 */

    xsh.x = 362;
    xsh.y = 282;
    xsh.width = 400;
    xsh.height = 300;
    xsh.flags = PPosition | PSize;

    XSetNormalHints(dpy, window1, &xsh);
}
    .
    .
    .
```

The example sets hints about the size and location of window *window1*.

3.5.2 Exchanging Properties Between Clients

Xlib provides routines that enable clients to exchange properties. The properties, which are global to the server, are called **selections**. Text cut from one window and pasted into another window exemplifies the global exchange of properties. The text cut in window A is a property owned by client A. Ownership of the property transfers to client B, who then pastes the text into window B.

Properties are exchanged between clients by a series of calls to routines that manage the selected text. When a user drags the pointer cursor, client A responds by calling the SET SELECTION OWNER routine. SET SELECTION OWNER identifies client A as the owner of the selected text. The routine also identifies the window of the selection, associates an atom with the text, and puts a timestamp on the selection. The atom, XA_PRIMARY, names the selection. The timestamp enables any clients competing for the selection to determine selection ownership.

Clients can determine the owner of a selection by calling the GET SELECTION OWNER routine.

When a user decides to paste the selected text in window B, client B, who owns window B, sends client A a selection request. The request identifies the window requesting the cut text and the format in which the client would like the property transferred.

In response to the request, client A first checks to ensure that the time of the request corresponds to the time in which client A owns the selection. If the time coincides, and if the selection is in the data type required by client B, client A notifies client B that the text is stored and available. The text is then moved to client B.

After receiving the text, client B informs client A that client B is the current owner of the selection.

In addition to requesting a selection in its current format, clients can call the CONVERT SELECTION routine. CONVERT SELECTION asks the owner of a selection to convert it to a particular data type. If conversion is possible, the client converting the selection notifies the client requesting the conversion that the selection is available. The property is then exchanged as previously described.

Clients request and notify other clients of selections by using events. For information about using events to request, convert, and notify clients of selections, see Chapter 9. For style guidelines about using selections, see the *XUI Style Guide*.

3.6 Changing Window Characteristics

Xlib provides routines that enable clients to change window position, size, border width, stacking order, and attributes.

This section describes how to use Xlib routines to do the following:

- Change multiple window characteristics in one call
- Change position, size, or border width
- Change stacking order
- Change window attributes

3.6.1 Reconfiguring Windows

Xlib enables clients either to change window characteristics using one call or to use individual routines to reposition, resize, or to change border width.

The CONFIGURE WINDOW routine enables clients to change window position, size, border width, and place in the hierarchy. To change these window characteristics in one call, use the CONFIGURE WINDOW routine, as follows:

1 Set values of relevant members of a window changes data structure.

2 Indicate what is to be reconfigured by specifying the appropriate flag in the CONFIGURE WINDOW **value_mask** argument.

The window changes data structure enables clients to specify one or more values for reconfiguring a window. The following illustrates the window changes data structure:

```
typedef struct {
    int x, y;
    int width, height;
    int border_width;
    Window sibling;
    int stack_mode;
} XWindowChanges;
```

Table 3–11 describes the members of the data structure.

Table 3–11 Window Changes Data Structure Members

Member Name	Contents
x	Defines, with the y member, the new location of the window relative to the origin of its parent.
y	Defines, with the x member, the new location of the window relative to the origin of its parent.
width	Defines the new width of the window, excluding the border.
height	Defines the new height of the window, excluding the border.
border_width	Specifies the new window border in pixels.
sibling	Specifies the sibling window for stacking order.
stack_mode	Defines how the window is restacked. Table 3–12 lists constants and definitions for restacking windows.

The client can change the hierarchical position of a window in relation to all windows in the stack or to a specified sibling. If the client changes the size, position, and stacking order of the window by calling CONFIGURE WINDOW, the server restacks the window based on its final, not initial, size and position. Table 3–12 lists constants and definitions for restacking windows.

Table 3–12 Stacking Values

Constants	Relative to All Windows	Relative to Siblings
Above	Top of stack.	Just above the sibling.
Below	Bottom of stack.	Just below the sibling.
TopIf	If any sibling obscures a window, the server places the obscured window on top of the stack.	If the specified sibling obscures a window, the server places the obscured window at the top of the stack.
BottomIf	If a window obscures any sibling, the server places the obscuring window at the bottom of the stack.	If a window obscures the specified sibling, the server places the obscuring window at the bottom of the stack.
Opposite	If any sibling obscures a window, the server places the obscured window on top of the stack. If a window obscures any window, the server places the obscuring window at the bottom of the stack.	If the specified sibling obscures a window, the server places the obscuring window on top of the stack. If a window obscures the specified sibling, the server places the obscuring window on the bottom of the stack.

Xlib assigns a symbol to the flag associated with each member of the data structure (Table 3–13).

Table 3–13 Window Changes Data Structure Flags

Flag Name	Window Changes Member
CWX	x
CWY	y
CWWidth	width
CWHeight	height
CWBorderWidth	border_width
CWSibling	sibling
CWStackMode	stack_mode

Example 3–5 illustrates using CONFIGURE WINDOW to change the position, size, and stacking order of a window when the user presses a button.

Example 3–5 Reconfiguring a Window Using the CONFIGURE WINDOW Routine

```
    /* This program changes the position, size, and stacking
       order of subwindow1 */

static void doButtonPress(eventP)
XEvent *eventP
{
    XWindowChanges xwc;

❶   xwc.x = 200;
    xwc.y = 350;
    xwc.width = 200;
    xwc.height = 50;
    xwc.sibling = subwindow2;
    xwc.stack_mode = Above;

❷   XConfigureWindow(dpy, subwindow1, CWX | CWY | CWWidth | CWHeight | CWSibling
            | CWStackMode, &xwc);
}
```

❶ Assign values to relevant members of the window changes data structure. Because the client identifies a sibling (*subwindow1*), it must also choose a mode for stacking operations.

❷ The call to reconfigure *subwindow1*. The CONFIGURE WINDOW routine call has the following format:

```
XConfigureWindow(display, window_id, change_mask, values)
```

Create a mask by performing a bitwise OR operation on relevant flags that indicate which members of WINDOW CHANGES the client has defined.

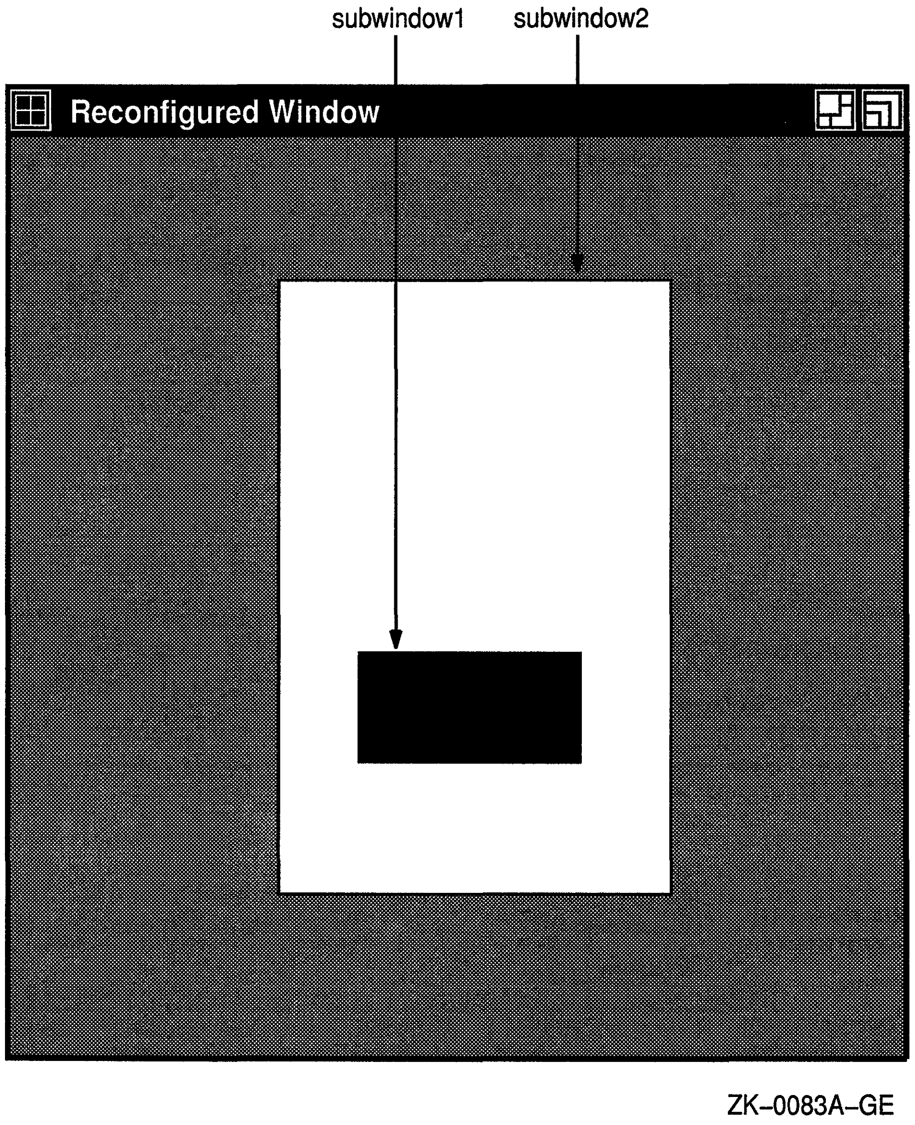
Figure 3–7 illustrates how the windows look after being reconfigured.

Table 3–14 lists routines to change individual window characteristics.

Table 3–14 Window Configuration Routines

Routine	Description
MOVE WINDOW	Moves a window without changing its size.
RESIZE WINDOW	Changes the size of a window without moving it. The upper left window coordinate does not change after resizing.
MOVE RESIZE WINDOW	Moves and changes the size of a window.
SET WINDOW BORDER WIDTH	Changes the border width of a window.

Figure 3–7 Reconfigured Window



3.6.2 Effects of Reconfiguring Windows

It is important to know how reconfiguring windows affects graphics and text drawn in them by the client. (See Chapter 6 for a description of working with graphics and Chapter 8 for a description of writing text.) When a client resizes a window, window contents are either moved or lost, depending on the **bit gravity** of the window. Bit gravity indicates that a designated region of the window should be relocated when the window is resized. Resizing also causes the server to resize children of the changed window.

To control how the server moves children when a parent is resized, set the **window gravity** attribute. Table 3–15 lists choices for retaining window contents and controlling how the server relocates children.

Table 3–15 Gravity Definitions

Constant Name	Movement of Window Contents and Subwindows
ForgetGravity	The server always discards window contents and tiles the window with its selected background. If the client has not specified a background, existing screen contents remain the same.
NorthWestGravity	Not moved.
NorthGravity	Moved to the right half the window width.
NorthEastGravity	Moved to the right the distance of the window width.
WestGravity	Moved down half the window height.
CenterGravity	Moved to the right half the window width and down half the window height.
EastGravity	Moved to the right the distance of the window width and down half the window height.
SouthWestGravity	Moved down the distance of the window height.
SouthGravity	Moved to the right half the window width and down the distance of the window height.
SouthEastGravity	Moved to the right the distance of the window width and down the distance of the window height.
StaticGravity	Contents or origin not moved relative to the origin of the root window. Static gravity only takes effect with a change in window width and height.
UnmapGravity	Window should not be moved; the child should be unmapped when the parent is resized.

The client can change the hierarchical position of a window in relation to either all windows in the stack or to a specified sibling. If the client changes the size, position, and stacking order of the window by calling CONFIGURE WINDOW, the server restacks the window based on its final, not initial, size and position. Table 3–12 lists constants and definitions for restacking windows.

Figure 3–8 illustrates how the server moves the contents of a reconfigured window when the bit gravity is set to the constant **EastGravity**.

Figure 3–9 illustrates how the server moves a child window if its parent is resized and its window gravity is set to the constant **NorthwestGravity**.

Figure 3–8 East Bit Gravity

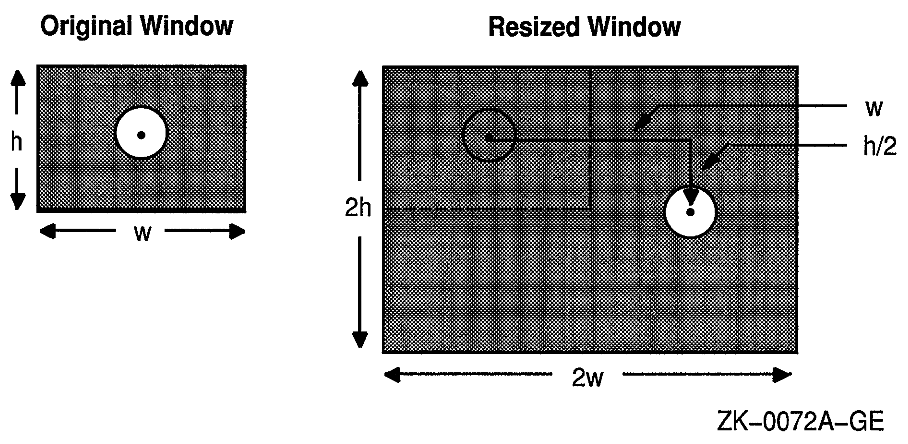
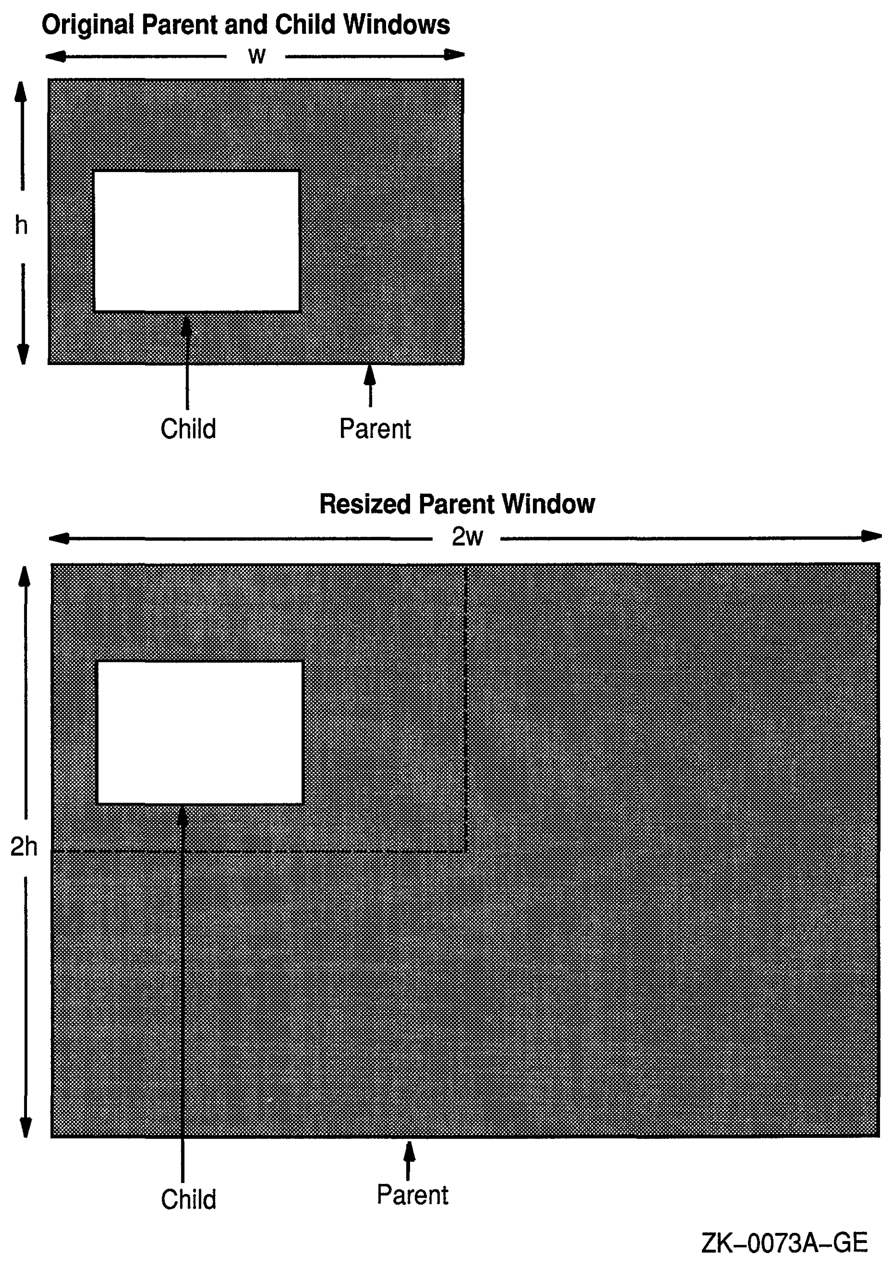


Figure 3–9 Northwest Window Gravity



3.6.3 Changing Stacking Order

Xlib provides routines that alter the window stacking order in the following ways:

- A specified window moves to either the top or the bottom of the stack.

- The lowest mapped child obscured by a sibling moves to the top of the stack.
- The highest mapped child that obscures a sibling moves to the bottom of the stack.

Use the RAISE WINDOW and LOWER WINDOW routines to move a specified window to either the top or the bottom of the stack, respectively.

To raise the lowest mapped child of an obscured window to the top of the stack, call CIRCULATE SUBWINDOWS UP. To lower the highest mapped child that obscures another child, call CIRCULATE SUBWINDOWS DOWN. The CIRCULATE SUBWINDOWS routine enables the client to perform these operations by specifying either the constant **RaiseLowest** or the constant **LowerHighest**.

To change the order of the window stack, use RESTACK WINDOW, which changes the window stack to a specified order. Reordered windows must have a common parent. If the first window the client specifies has other unspecified siblings, its order relative to those siblings remains unchanged.

3.6.4 Changing Window Attributes

Xlib provides routines that enable clients to change the following:

- Default contents of an input-output window
- Border of an input-output window
- Treatment of the window when it or its relative is obscured
- Treatment of the window when it or its relative is moved
- Information the window receives about operations associated with other windows
- Color
- Cursor

Section 3.2.2 includes descriptions of window attributes and their relationship to the set window attributes data structure.

This section describes how to change any attribute using the CHANGE WINDOW ATTRIBUTES routine. In addition to CHANGE WINDOW ATTRIBUTES, Xlib includes routines that enable clients to change background and border attributes. Table 3–16 lists these routines and their functions.

Table 3–16 Routines for Changing Window Attributes

Routine	Description
SET WINDOW BACKGROUND	Sets the background pixel
SET WINDOW BACKGROUND PIXMAP	Sets the background pixmap
SET WINDOW BORDER	Sets the window border to a specified pixel
SET WINDOW BORDER PIXMAP	Sets the window border to a specified pixmap

To change any window attribute, use CHANGE WINDOW ATTRIBUTES as follows:

1 Assign a value to the relevant member of a set window attributes data structure.

2 Indicate the attribute to change by specifying the appropriate flag in the CHANGE WINDOW ATTRIBUTES **value_mask** argument. To define more than one attribute, indicate the attributes by doing a bitwise OR on the appropriate flags.

See Table 3–3 for symbols Xlib assigns to each member to facilitate referring to the attributes.

Example 3–6 illustrates using CHANGE WINDOW ATTRIBUTES to redefine the characteristics of a window.

Example 3–6 Changing Window Attributes

```
    XSetWindowAttributes  xswa;

❶   xswa.background_pixel = BlackPixelOfScreen(dpy);
    xswa.border_pixel = WhitePixelOfScreen(dpy);

❷   XChangeWindowAttributes(dpy, win, CWBorderPixel | CWBackPixel, &xswa);
    .
    .
    .
```

❶ Assign new values to a set window attributes data structure.

❷ Call CHANGE WINDOW ATTRIBUTES to change the window attributes. The CHANGE WINDOWS ATTRIBUTES routine has the following format:

```
XChangeWindowAttributes(display, window_id,
        attributes_mask, attributes)
```

Specify the attributes to change with a bitwise inclusive OR of the relevant symbols listed in Table 3–3. The **values** argument passes the address of a set window attributes data structure.

Table 3–17 lists changes in attributes and their effects.

Table 3–17 Effects of Window Attribute Changes

Attribute Changed	Effects
Background	Window contents are unchanged.
	If the window is a root window, specifying the constant None or ParentRelative restores the default background pixmap.
	The server does not repaint the background automatically.
Border	Setting the border causes the border to be repainted.
	If a background change causes a change in the border tile origin, the server repaints the border.
	Specifying the constant CopyFromParent on a root window restores the default border pixmap.
Bit and window gravity	A change in window gravity has no effect until the window is resized.
Backing store	In this release, backing store is not supported.
Backing planes	In this release, backing planes is not supported.
Backing pixels	In this release, backing pixels is not supported.
Save under	If the window is mapped, changing the value of save under may have no immediate effect.
Event mask	See Chapter 9.
Do not propagate mask	See Chapter 9.
Color map	See Chapter 5.
Cursor	Specifying the constant None on a root window restores the default cursor.

3.7 Getting Information About Windows

Using Xlib information routines, clients can get information about the parent, children, and number of children in a window tree; window geometry; the root window in which the pointer is currently visible; and window attributes.

Table 3–18 lists and describes Xlib routines that return information about windows.

Table 3–18 Window Information Routines

Routine	Description
QUERY TREE	Returns information about the window tree
GET GEOMETRY	Returns information about the root window identifier, coordinates, width and height, border width, and depth
QUERY POINTER	Returns the root window the pointer is currently on and the pointer coordinates relative to the root window origin
GET WINDOW ATTRIBUTES	Returns information from the window attributes data structure

To get information about window attributes, use the GET WINDOW ATTRIBUTES routine. The client receives requested information in the window attributes data structure. The following illustrates the window attributes data structure:

```
typedef struct {
    int x, y;
    int width, height;
    int border_width;
    int depth;
    Visual *visual;
    Window root;
    int class;
    int bit_gravity;
    int win_gravity;
    int backing_store;
    unsigned long backing_planes;
    unsigned long backing_pixel;
    Bool save_under;
    Colormap colormap;
    Bool map_installed;
    int map_state;
    long all_event_masks;
    long your_event_mask;
    long do_not_propagate_mask;
    Bool override_redirect;
} XWindowAttributes;
```

Table 3–19 describes the members of the window attributes data structure.

Table 3–19 Window Attributes Data Structure Members

Member Name	Contents
x	Specifies, with the y member, the coordinates of the upper left corner of the window relative to its parent.
y	Specifies, with the x member, the coordinates of the upper left corner of the window relative to its parent.
width	Specifies the width of the window, excluding the window border, in pixels.
height	Specifies the height of the window, excluding the window border, in pixels.
border_width	Specifies the width of the window border in pixels.
depth	Specifies the bits per pixel of the window.
visual	A pointer to a visual data structure associated with the window. The visual data structure specifies how displays should treat color resources. For more information, see Section 3.5.1.
root	Identifies the screen with which the window is associated.
class	Specifies whether the window accepts input and output, or input only.
bit_gravity	Specifies how pixels should be moved when the window is resized.
win_gravity	Specifies how the window should be repositioned when its parent is resized.
backing_store	Indicates whether or not the server should maintain a record of portions of a window that are obscured when the window is mapped. In this release, clients must maintain contents of obscured windows.
backing_planes	Indicates (with bits set to 1) which bit planes of the window hold dynamic data that must be preserved in backing stores and during save under operations. In this release, clients must maintain data to be preserved.
backing_pixel	Defines what values to use in planes not specified by the backing_planes member. In this release, clients must maintain values to be saved.
save_under	Setting the save_under member to true informs the server that the client would like the contents of the screen saved when the window obscures them. Saving the contents of obscured portions of the screen is not guaranteed.
color map	Specifies the color map, if any, that best reflects the colors of the window. The color map must have the same visual type as the window. If it does not, an error occurs. For more information about color maps, see Chapter 5.
map_installed	If set to true, the map_installed member indicates that the color map is currently installed and that the window is being displayed in its correct colors.

(continued on next page)

Table 3–19 (Cont.) Window Attributes Data Structure Members

Member Name	Contents
map_state	Indicates whether the window is mapped and viewable. Clients can specify the following constants:

Constant Name	Description
IsUnmapped	Indicates that the window is not mapped
IsUnviewable	Indicates that the window is mapped, but that one of its ancestors is unmapped, causing the window to be unviewable
IsViewable	Indicates that the window is mapped and viewable

all_events_mask	Indicates the set of events in which all applications have an interest. The all_events_ mask member is the inclusive OR of event masks set for the window. For more information about event masks, see Chapter 9.
your_event_mask	Indicates the events about which the querying application is interested in receiving notice.
do_not_propagate	Defines which events should not be propagated to the window's ancestors when no application has the event type selected in the window.
override_redirect	Specifies whether requests to map and configure the window should override a request by another client to redirect those calls (see Chapter 9). Typically, this mask, which informs the window manager not to tamper with the window, should be used only on subwindows such as menus.
screen	Specifies the screen on which the window is mapped.

4 Defining Graphics Characteristics

After opening a display and creating a window, clients can draw lines and shapes, create cursors, and draw text. Creating a graphics object is a two-step process. Clients first define the characteristics of the graphics object and then create it. For example, before creating a line, a client first defines line width and style. After defining the characteristics, the client creates the line with the specified width and style.

This chapter describes how to define the graphics characteristics prior to creating them, including the following topics:

- The graphics context—A description of the graphics characteristics a client can define and the GC values data structure used to define them
- Defining graphics characteristics—How to define graphics characteristics using the CREATE GC routine
- Copying, changing, and freeing attributes—How to copy, change, and undefine graphics characteristics
- Defining graphics characteristics efficiently—How to work efficiently with several sets of graphics characteristics

Chapter 6 describes how to create graphics objects. Chapter 8 describes how to work with text.

4.1 The Graphics Context

The characteristics of a graphics object make up its **graphics context**. As with window characteristics, Xlib provides a data structure and routine to enable clients to define multiple graphics characteristics easily. By setting values in the GC values data structure and calling the CREATE GC routine, clients can define all characteristics relevant to a graphics object.

Xlib also provides routines that enable clients to define individual or functional groups of graphics characteristics.

Xlib always records the defined values in a GC data structure, which is reserved for the use of Xlib and the server only. This occurs when clients define graphic characteristics using either the CREATE GC routine or one of the individual routines. Table 4–1 lists the default values of the GC data structure.

Table 4–1 GC Data Structure Default Values

Member	Default Value
Function	GXcopy
Plane mask	All ones
Foreground	0
Background	1
Line width	0
Line style	Solid
Cap style	Butt
Join style	Mitre
Fill style	Solid
Fill rule	Even odd
Arc mode	Pie slice
Tile	Pixmap of unspecified size filled with foreground pixel
Stipple	Pixmap of unspecified size filled with ones
Tile or stipple x origin	0
Tile or stipple y origin	0
Font	Varies with implementation
Subwindow mode	Clip by children
Graphics exposures	True
Clip x origin	0
Clip y origin	0
Clip mask	None
Dash offset	0
Dashes	4 (the list [4,4])

4.2 Defining Multiple Graphics Characteristics in One Call

Xlib enables clients to define multiple characteristics of a graphics object in one call. To define multiple characteristics, use the CREATE GC routine as follows:

1 Assign values to the relevant members of the GC values data structure.

2 Indicate the attributes to define by specifying the appropriate flag in the **value_mask** argument of the routine. To define more than one attribute, do a bitwise OR on the appropriate attribute flags.

The following illustrates the GC values data structure:

```
typedef struct {
        int function;
        unsigned long plane_mask;
        unsigned long foreground;
        unsigned long background;
        int line_width;
        int line_style;
        int cap_style;
        int join_style;
        int fill_style;
        int fill_rule;
        int arc_mode;
        Pixmap tile;
        Pixmap stipple;
        int ts_x_origin;
        int ts_y_origin;
        Font font;
        int subwindow_mode;
        Bool graphics_exposures;
        int clip_x_origin;
        int clip_y_origin;
        Pixmap clip_mask;
        int dash_offset;
        char dashes;
} XGCValues;
```

Table 4–2 describes the members of the data structure.

Table 4–2 GC Values Data Structure Members

Member Name	Contents
function	Defines how the server computes pixel values when the client updates a section of the screen. The following lists available functions:

Constant Name	Description
GXclear	0
GXand	src AND dst
GXandReverse	src AND NOT dst
GXcopy	src
GXandInverted	(NOT src) AND dst
GXnoop	dst
GXxor	src XOR dst
GXor	src OR dst
GXnor	(NOT src) AND NOT dst
GXequiv	(NOT src) XOR dst
GXinvert	NOT dst
GXorReverse	src OR NOT dst
GXcopyInverted	NOT src
GXorInverted	(NOT src) OR dst
GXnand	(NOT src) OR NOT dst
GXset	1

Member Name	Contents
	The screen the client is updating is the destination (dst). The graphics context the client uses to update the screen is the source (src). The function member specifies how the server computes new destination bits from the source (src) and the old bits of the destination (dst). The most common logical function is the default specified by the constant GXcopy, which only uses relevant values in the specified GC values data structure to update the screen.
plane_mask	Specifies the planes on which the server performs the bitwise computation of pixels, defined by the function member. Because a monochrome display has only one plane, the plane mask value is given in the least significant bit of the longword. As planes are added to the display hardware, they are defined in the more significant bits of the mask. The display routine ALL PLANES specifies that all planes of the display are referred to simultaneously. The server does not perform range checking on the plane mask. It truncates values to the appropriate number of bits.
foreground	Specifies an index to a color map entry for foreground color.
background	Specifies an index to a color map entry for background color.

(continued on next page)

Table 4–2 (Cont.) GC Values Data Structure Members

Member Name	Contents

line_width	Defines the width of a line in pixels.
	
	The server draws a line with a width of one or more pixels centered on the path described in the graphics request and contained within a bounding box. Unless otherwise specified by the join or cap style, the bounding box of a line with endpoints [$x1$, $y1$], [$x2$, $y2$] and width $w > 0$ is a rectangle with vertices at the following real coordinates:
	
	`[x1-w*sn/2, y1+w*cs/2], [x1+w*sn/2, y1-w*cs/2]`
	`[x2-w*sn/2, y2+w*cs/2], [x2+w*sn/2, y2-w*cs/2]`
	
	In this example, sn is the sine of the angle of the line. The symbol cs is the cosine of the angle of the line. A pixel is part of the line and is drawn if the center of the pixel is fully inside the bounding box. If the center of the pixel is exactly on the bounding box, the pixel is part of the line if and only if the interior is immediately to its right (x increasing direction). Pixels with centers on a horizontal edge are a special case and are part of the line if and only if the interior is immediately below the bounding box (y increasing direction). See Figure 4–1.
	
	Lines with zero line width are one pixel wide. The server draws them using an unspecified, device-dependent algorithm that imposes the following two constraints:
	
	• If the server draws the line unclipped from [$x1$, $y1$] to [$x2$, $y2$], and if the server draws a second line from [$x1 + dx$, $y1 + dy$] to [$x2 + dx$, $y2 + dy$], then point [x, y] is touched by drawing the first line if and only if the point [$x + dx$, $y + dy$] is touched by drawing the second line.
	• The effective set of points that compose a line cannot be affected by clipping. That is, a point is touched in a clipped line if and only if the point lies inside the clipping region and if the point would be touched by the line when drawn unclipped.
	
	A line more than one pixel wide drawn from [$x1$, $y1$] to [$x2$, $y2$] always draws the same pixels as a line of the same width drawn from [$x2$, $y2$] to [$x1$, $y1$], excluding cap and join styles.
	
	In general, drawing a line whose line width is zero is substantially faster than drawing a line whose line width is one or more. However, because the drawing algorithms for thin lines is different than those for wide lines, thin lines may not look as good when mixed with wide lines. If clients want precise and uniform results across all displays, they should always use a line width of one or more. Note, however, that specifying a line width of greater than zero decreases performance substantially.

(continued on next page)

Table 4–2 (Cont.) GC Values Data Structure Members

Member Name	Contents
line_style	Defines which sections of the line the server draws. The following lists available line styles and the constants that specify them:
	Constant Name: LineSolid — The full path of the line is drawn.
	LineDoubleDash — The full path of the line is drawn, but the even dashes are filled differently than the odd dashes, with cap butt style used where even and odd dashes meet.
	LineOffOnDash — Only the even dashes are drawn. The cap_style member applies to all internal ends of dashes. Specifying the constant CapNotLast is equivalent to specifying CapButt.
	Figure 4–2 illustrates the styles.
cap_style	Defines how the server draws the endpoints of a path. The following lists available cap styles and the constants that specify them:
	Constant Name: CapButt — Square at the endpoint (perpendicular to the slope of the line) with no projection beyond the endpoint.
	CapNotLast — Equivalent to CapButt, except that the final endpoint is not drawn if the line width is zero or one.
	CapRound — A circular arc with the diameter equal to the line width, centered on the endpoint (equivalent to the value specified by CapButt for a line width of zero or one).
	CapProjecting — Square at the end, but the path continues beyond the endpoint for a distance equal to half the width of the line (equivalent to the value specified by the constant CapButt for a line width of zero or one).
	Figure 4–3 illustrates the butt, round, and projecting cap styles. Figure 4–4 illustrates the style specified by the constant CapNotLast.

Nested tables:

Constant Name	Description
LineSolid	The full path of the line is drawn.
LineDoubleDash	The full path of the line is drawn, but the even dashes are filled differently than the odd dashes, with cap butt style used where even and odd dashes meet.
LineOffOnDash	Only the even dashes are drawn. The cap_style member applies to all internal ends of dashes. Specifying the constant CapNotLast is equivalent to specifying CapButt.

Constant Name	Description
CapButt	Square at the endpoint (perpendicular to the slope of the line) with no projection beyond the endpoint.
CapNotLast	Equivalent to CapButt, except that the final endpoint is not drawn if the line width is zero or one.
CapRound	A circular arc with the diameter equal to the line width, centered on the endpoint (equivalent to the value specified by CapButt for a line width of zero or one).
CapProjecting	Square at the end, but the path continues beyond the endpoint for a distance equal to half the width of the line (equivalent to the value specified by the constant CapButt for a line width of zero or one).

(continued on next page)

Table 4–2 (Cont.) GC Values Data Structure Members

Member Name	Contents
	If a line has coincident endpoints ($x1 = x2$, $y1 = y2$), the cap style is applied to both endpoints with the following results:

Constant Name	Line Width	Description
CapNotLast	Thin	Device dependent, but the desired effect is that nothing is drawn
CapButt	Thin	Device dependent, but the desired effect is that a single pixel is drawn
CapButt	Wide	Nothing is drawn
CapRound	Thin	Device dependent, but the desired effect is that a single pixel is drawn
CapRound	Wide	The closed path is a circle, centered at the endpoint, with the diameter equal to the line width
CapProjecting	Thin	Device dependent, but the desired effect is that a single pixel is drawn
CapProjecting	Wide	The closed path is a square, aligned with the coordinate axes, centered at the endpoint with sides equal to the line width

Member Name	Contents
join_style	Defines how the server draws corners for wide lines. Available join styles and the constants that specify them are as follows:

Constant Name	Description
JoinMitre	The outer edges of the two lines extend to meet at an angle
JoinRound	A circular arc with diameter equal to the line width, centered at the join point
JoinBevel	Cap butt endpoint style, with the triangular notch filled

Figure 4–5 illustrates the styles.

For a line with coincident endpoints ($x1 = x2$, $y1 = y2$), when the join style is applied at one or both endpoints, the effect is as if the line were removed from the overall path. However, if the total path consists of (or is reduced to) a single point joined with itself, the effect is the same as if the cap style were applied to both endpoints.

(continued on next page)

Table 4–2 (Cont.) GC Values Data Structure Members

Member Name	Contents
fill_style	Specifies the contents of the source for line, text, and fill operations. The following lists available fill styles for text and fill requests (DRAW TEXT, DRAW TEXT 16, FILL RECTANGLE, FILL POLYGON, FILL ARC). It also lists available styles applicable to solid lines and even dashes resulting from line requests (LINE, SEGMENTS, RECTANGLE, ARC):

Constant Name	Description
FillSolid	Foreground
FillTiled	Tile
FillOpaqueStippled	A tile with the same width and height as stipple but with background everywhere stipple has a zero and with foreground everywhere stipple has a one
FillStippled	Foreground masked by stipple

The following lists available styles applicable to odd dashes resulting from line requests:

Constant Name	Description
FillSolid	Background
FillTiled	Tile
FillOpaqueStippled	A tile with the same width and height as stipple but with background everywhere stipple has a zero and with foreground everywhere stipple has a one
FillStippled	Background masked by stipple

Member Name	Contents
fill_rule	Defines what pixels the server draws along a path when a polygon is filled (see Section 6.5.2). The two available choices are EvenOddRule and WindingRule. The EvenOddRule constant defines a point to be inside a polygon if an infinite ray with the point as origin crosses the path an odd number of times. If the point meets these conditions, the server draws a corresponding pixel.

(continued on next page)

Table 4–2 (Cont.) GC Values Data Structure Members

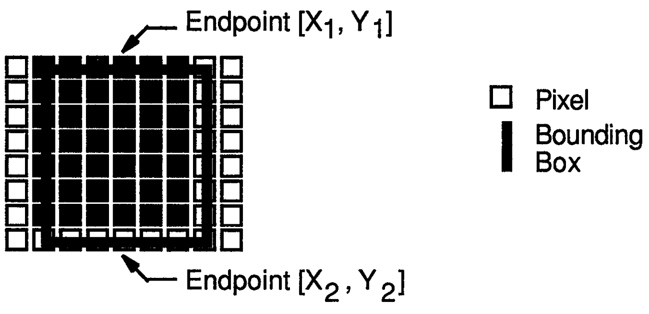
Member Name	Contents
	The WindingRule constant defines a point to be inside the polygon if an infinite ray with the pixel as origin crosses an unequal number of clockwise-directed and counterclockwise-directed path segments. A clockwise-directed path segment is one that crosses the ray from left to right as observed from the pixel. A counterclockwise-directed segment is one that crosses the ray from right to left as observed from that point. When a directed line segment coincides with a ray, choose a different ray that is not coincident with a segment. If the point meets these conditions, the server draws a corresponding pixel. For both even odd rule and winding rule, a point is infinitely small, and the path is an infinitely thin line. A pixel is inside the polygon if the center point of the pixel is inside, and the center point is not on the boundary. If the center point is on the boundary, the pixel is inside if and only if the polygon interior is immediately to its right (x increasing direction). Pixels with centers along a horizontal edge are a special case and are inside if and only if the polygon interior is immediately below (y increasing direction). Figure 4–6 illustrates fill rules. Figure 4–7 illustrates rules for filling a pixel when it falls on a boundary.
arc_mode	Controls how the server fills an arc. The available choices are the values specified by the ArcPieSlice and ArcChord constants. Figure 4–8 illustrates the two modes.
tile	Specifies the pixmap the server uses for tiling operations. The pixmap must have the same root and depth as the graphics context, or an error occurs. Clients can use any size pixmap for tiling, although some sizes produce a faster response than others. To determine the optimum size, use the QUERY BEST SIZE routine. Storing a pixmap in a graphics context might or might not result in a copy being made. If the pixmap is used later as the destination for a graphics request, the change might or might not be reflected in the graphics context. If the pixmap is used simultaneously in a graphics request both as a destination and as a tile, the results are not defined.
stipple	Specifies the pixmap the server uses for stipple operations. The pixmap must have the same root as the graphics context and a depth of one, or an error occurs. For stipple operations where the fill style is specified as the FillStippled constant but not the FillOpaqueStipple constant, the stipple pattern is tiled in a single plane and acts as an additional clip mask. Perform a bitwise AND operation with the clip mask. Clients can use any size pixmap for stipple operations, although some sizes produce a faster response than others. To determine the optimum size, use the QUERY BEST SIZE routine. Storing a pixmap in a graphics context might or might not result in a copy being made. If the pixmap is used later as the destination for a graphics request, the change might or might not be reflected in the graphics context. If the pixmap is used simultaneously in a graphics request both as a destination and as a stipple, the results are not defined.

(continued on next page)

Table 4–2 (Cont.) GC Values Data Structure Members

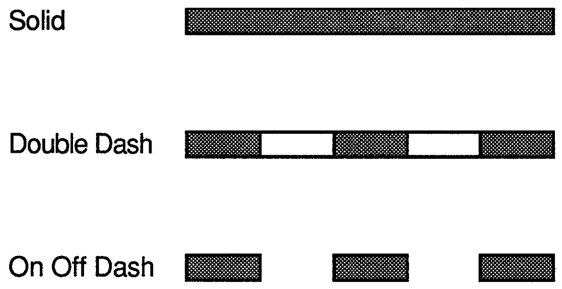
Member Name	Contents
ts_x_origin	Defines the origin for tiling and stipple operations. Origins are relative to the origin of whatever window or pixmap is specified in the graphics request.
ts_y_origin	Defines the origin for tiling and stipple operations. Origins are relative to the origin of whatever window or pixmap is specified in the graphics request.
font	Specifies the font that the server uses for text operations.
subwindow_mode	Specifies whether or not inferior windows clip superior windows. The constant ClipByChildren specifies that all viewable input-output children clip both source and destination windows. The constant IncludeInferiors specifies that inferiors clip neither source nor destination windows. This results in drawing through subwindow boundaries. The semantics of using the constant on a window with a depth of one and with mapped inferiors of differing depth is undefined by the core protocol.
graphic_exposures	Specifies whether or not the server informs the client when the contents of a window region are lost.
clip_x_origin	Defines the x-coordinate of the clip origin. The clip origin specifies the point within the clip region that is aligned with the drawable origin.
clip_y_origin	Defines the y-coordinate of the clip origin. The clip origin specifies the point within the clip region that is aligned with the drawable origin.
clip_mask	Identifies the pixmap the server uses to restrict write operations to the destination drawable. The pixmap must have a depth of one and have the same root as the graphics context. The clip mask clips only the destination drawable, not the source drawable. Where a value of one appears in the mask, the corresponding pixel in the destination drawable is drawn; where a value of zero occurs, no pixel is drawn. Any pixel within the destination drawable that is not represented within the clip mask pixmap is not drawn. When a client specifies the value of clip mask as None, the server draws all pixels.
dash_offset	Specifies the pixel within the dash length sequence, defined by the dashes member, to start drawing a dashed line. For example, a dash offset of zero starts a dashed line as the beginning of the dash line sequence. A dash offset of five starts the line at the fifth pixel of the line sequence. Figure 4–9 illustrates dashed offsets.
dashes	Specifies the length, in number of pixels, of each dash. The value of this member must be nonzero or an error occurs.

Figure 4–1 Bounding Box



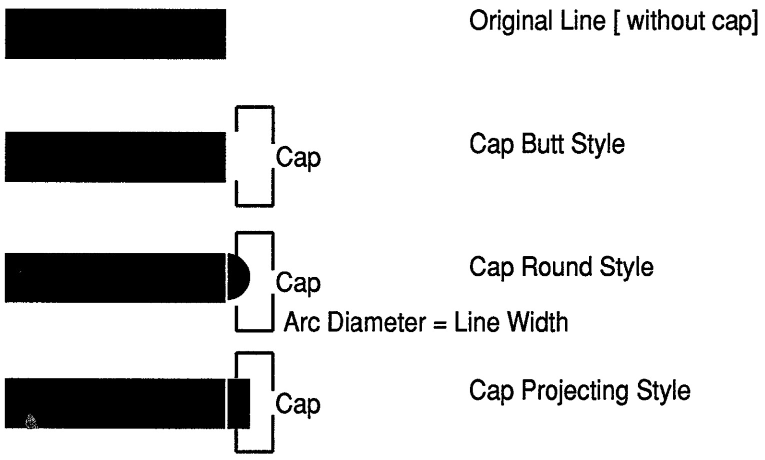
ZK-0011A-GE

Figure 4–2 Line Styles



ZK-0010A-GE

Figure 4–3 Butt, Round, and Projecting Cap Styles



ZK-0012A-GE

Figure 4–4 Cap Not Last Style

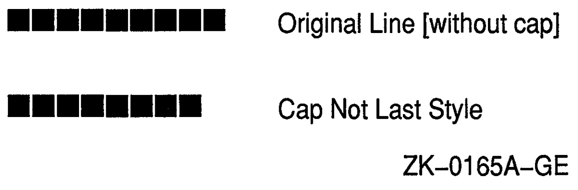


Figure 4–5 Join Styles

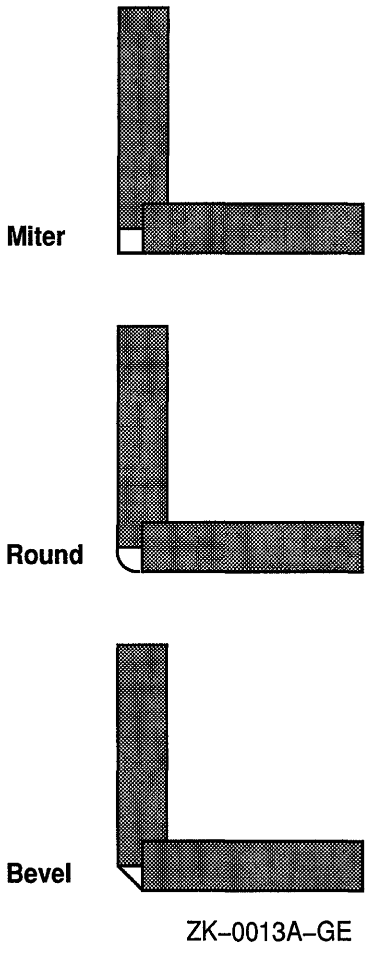


Figure 4–6 Fill Rules

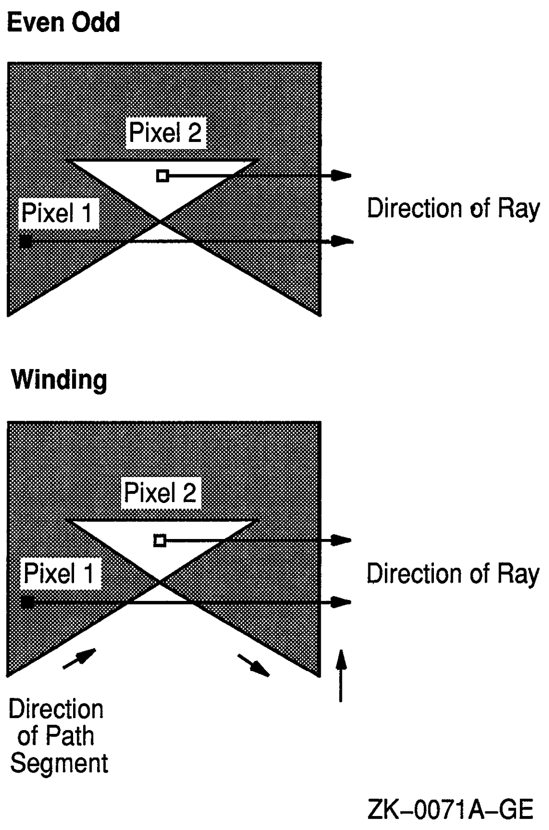


Figure 4–7 Pixel Boundary Cases

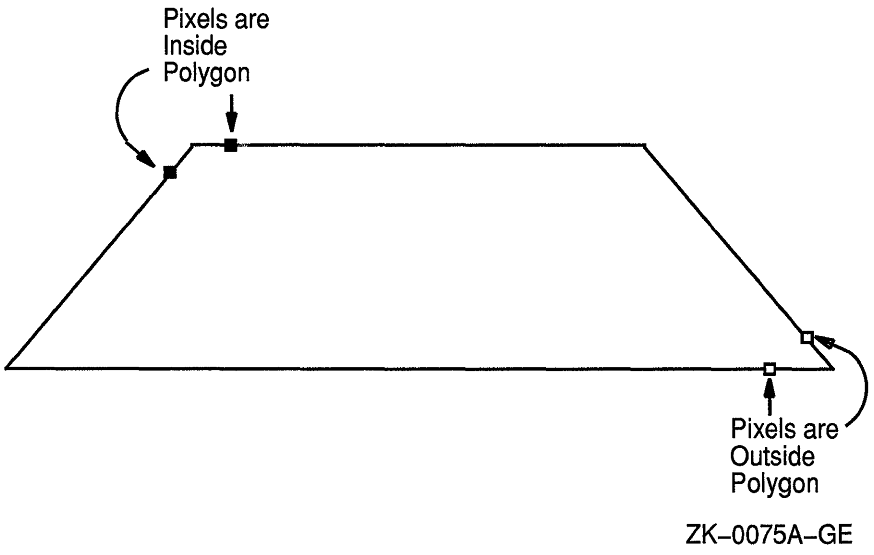


Figure 4–8 Styles for Filling Arcs

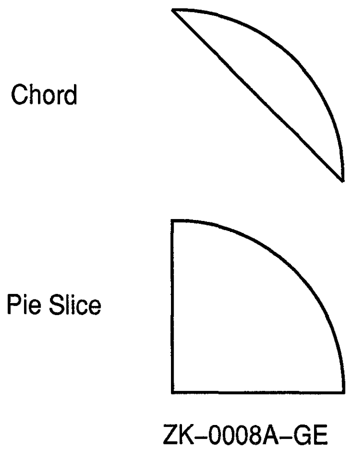
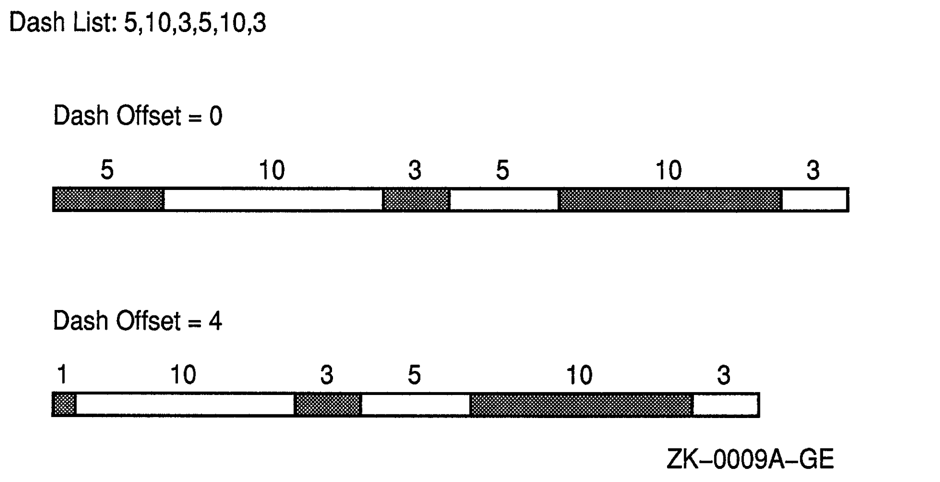


Figure 4–9 Dashed Line Offset



Xlib assigns a flag for each member of the GC values data structure to facilitate referring to members (Table 4–3).

Table 4–3 GC Values Data Structure Flags

Flag Name	GC Values Member
GCFunction	function
GCPlaneMask	plane_mask
GCForeground	foreground
GCBackground	background
GCLineWidth	line_width
GCLineStyle	line_style
GCCapStyle	cap_style
GCJoinStyle	join_style
GCFillStyle	fill_style
GCFillRule	fill_rule
GCTile	tile
GCStipple	stipple
GCTileStipXOrigin	ts_x_origin
GCTileStipYOrigin	ts_y_origin
GCFont	font
GCSubwindowMode	subwindow_mode
GCGraphicsExposures	graphics_exposures
GCClipXOrigin	clip_x_origin

(continued on next page)

Table 4–3 (Cont.) GC Values Data Structure Flags

Flag Name	GC Values Member
GCClipYOrigin	clip_y_origin
GCXClipMask	clip_mask
GCDashOffset	dash_offset
GCDashList	dash_list
GCArcMode	arc_mode

Example 4–1 illustrates how a client can define graphics context values using the CREATE GC routine. Figure 4–10 shows the resulting output.

Example 4–1 Defining Graphics Characteristics Using the CREATE GC Routine

```
    /* Create window win on                                  *
     *  display dpy, defined as follows:                     *
     *     Position: x = 100,y = 100                         *
     *     Width = 600                                       *
     *     Height = 600                                      *
     *  gc refers to the graphics context                    */
    .
    .
    .
❶GC gc;
    .
    .
    .
static void doCreateGraphicsContext( )
{
❷    XGCValues xgcv;

    /* Create graphics context. */

❸   xgcv.foreground = doDefineColor(3);
    xgcv.background = doDefineColor(4);
    xgcv.line_width = 4;
    xgcv.line_style = LineDoubleDash;
    xgcv.dash_offset = 0;
    xgcv.dashes = 25;

❹   gc = XCreateGC(dpy, win, GCForeground | GCBackground
         | GCLineWidth | GCLineStyle | GCDashOffset | GCDashList, &xgcv);
}
    .
    .
    .
static void doButtonPress(eventP)
XEvent *eventP;
{
    x1 = y1 = 100;
    x2 = y2 = 550;

❺   XDrawLine(dpy, win, gc, x1, y1, x2, y2);
}
```

❶ Assign storage for a graphics context (GC) data structure. The scope of *gc* is global to enable windowing and graphics routines in other modules to refer to it.

❷ Once the client defines characteristics with the GC values data structure, Xlib does not have to refer to the data structure again.

❸ Specify the foreground, background, line width, line style, dash offset, and dashes for line drawing.

The dashed line is four pixels wide. A dash offset value of zero starts dashes at the beginning of the line. The dashes value, referred to by GCDashList, specifies that dashes be 25 pixels long.

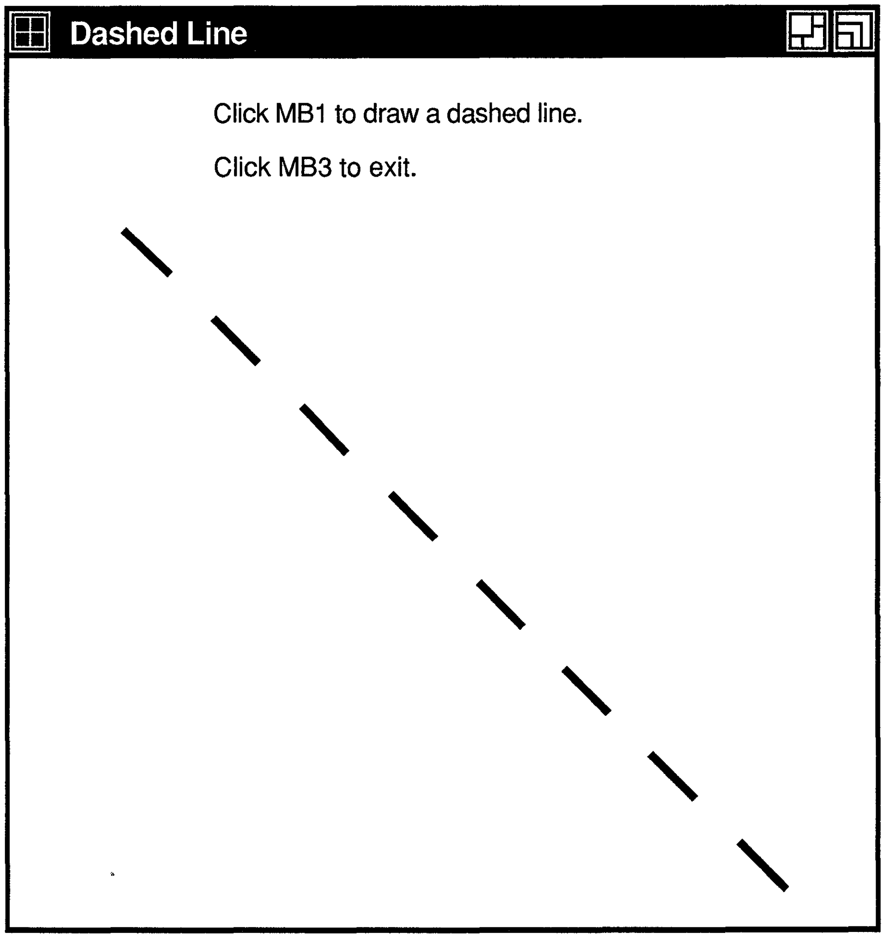
❹ The CREATE GC routine loads values into a GC data structure. The CREATE GC routine has the following format:

```
gc_id = XCreateGC (display, drawable_id, gc_mask,
          values_struc)
```

Indicate defined attributes with a bitwise OR that uses symbols listed in Table 4–3.

❺ See Chapter 6 for information about drawing lines.

Figure 4–10 Dashed Line



Click MB1 to draw a dashed line.

Click MB3 to exit.

ZK–0104A–GE

4.3 Defining Individual Graphics Characteristics

Xlib offers routines that enable clients to define individual or functional groups of graphics characteristics. Table 4–4 lists and briefly describes these routines. For more information about the components, see Section 4.1.

Table 4–4 Routines That Define Individual or Functional Groups of Graphics Characteristics

Routine	Description
Foreground, Background, Plane Mask, and Function Routines	
SET STATE	Sets the foreground, background, plane mask, and function
SET FOREGROUND	Sets the foreground
SET BACKGROUND	Sets the background
SET PLANE MASK	Sets the plane mask
SET FUNCTION	Sets the function
Line Attribute Routines	
SET LINE ATTRIBUTES	Sets line width, line style, cap style, and join style
SET LINE DASHES	Sets the dash offset and dash list of a line
Fill Style and Rule Routines	
SET FILL STYLE	Sets fill style to solid, tiled, stippled, or opaque stippled
SET FILL RULE	Sets fill rule to either even and odd or winding rule
Fill Tile and Stipple Routines	
QUERY BEST SIZE	Queries the server for the size closest to the one specified
QUERY BEST STIPPLE	Queries the server for the closest stipple shape to the one specified
QUERY BEST TILE	Queries the server for the closest tile shape to the one specified
SET STIPPLE	Sets the stipple pixmap
SET TILE	Sets the tile pixmap
SET TS ORIGIN	Sets the tile or stipple origin
Font Routine	
SET FONT	Sets the current font

(continued on next page)

Table 4–4 (Cont.) Routines That Define Individual or Functional Groups of Graphics Characteristics

Routine	Description
Clip Region Routines	
SET CLIP MASK	Sets the mask for bitmap clipping
SET CLIP ORIGIN	Sets the origin for clipping
SET CLIP RECTANGLES	Changes the clip mask from its current value to the specified rectangles
Arc, Subwindow, and Exposure Routines	
SET ARC MODE	Sets the arc mode to either chord or pie slice
SET SUBWINDOW MODE	Sets the subwindow mode to either clip by children or include inferiors
SET GRAPHICS EXPOSURES	Specifies whether exposure events are created when calling COPY AREA or COPY PLANE

Example 4–2 illustrates using individual routines to set background, foreground, and line attributes. Figure 4–11 illustrates the resulting output.

Example 4–2 Using Individual Routines to Define Graphics Characteristics

```
GC gc;
    .
    .
    .
static void doButtonPress(eventP)
XEvent *eventP;
{
❶   char dash_list[] = {20,5,10};
    x1 = y1 = 100;
    x2 = y2 = 550;

    XSetBackground(dpy, gc, doDefineColor(4));
❷   XSetLineAttributes(dpy, gc, 0, LineDoubleDash, 0, 0);
❸   XSetDashes(dpy, gc, 0, dash_list, 3);
    XDrawLine(dpy, win, gc, x1, y1, x2, y2);
}
```

❶ The *dash_list* variable defines the length of odd and even dashes. The first and third elements of the initialization list specify even dashes; the second element specifies odd dashes.

❷ The SET LINE ATTRIBUTES routine enables the client to define line width, style, cap style, and join style in one call.

The SET LINE ATTRIBUTES routine has the following format:

```
XSetLineAttributes(display, gc_id, line_width, line_style,
      cap_style, join_style)
```

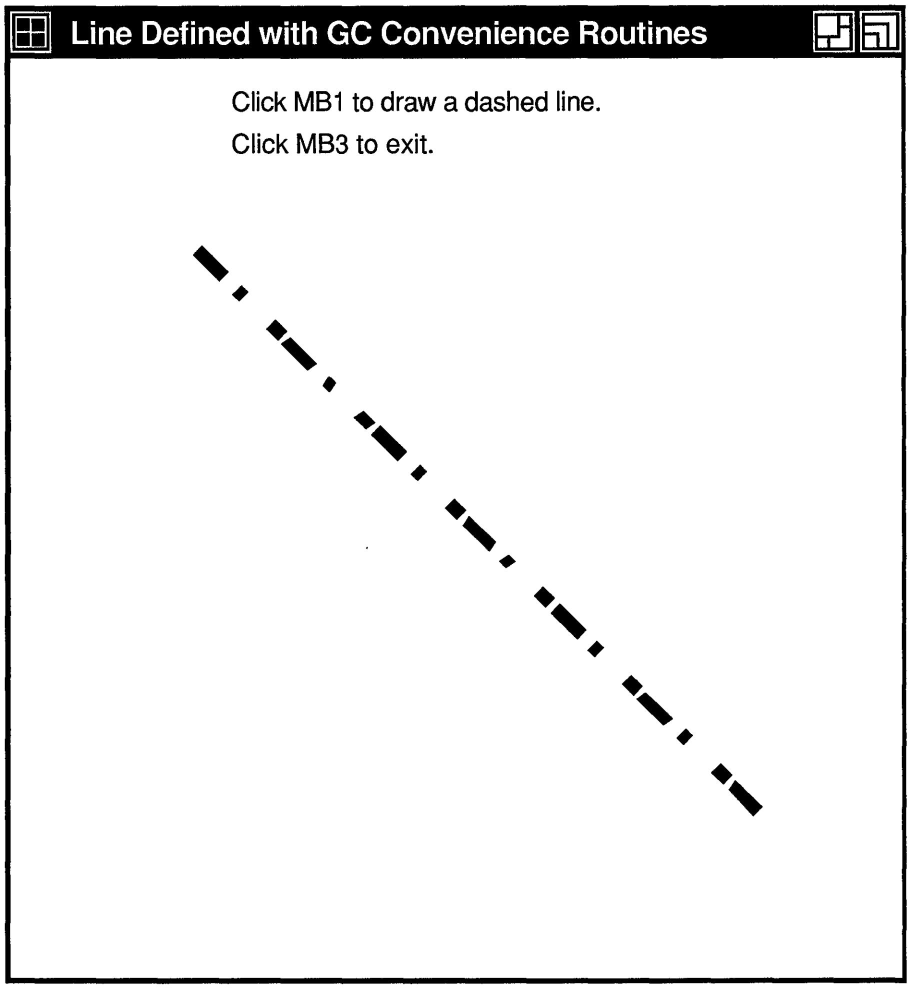
The zero **cap_style** argument specifies the default cap style.

❸ If using the CREATE GC routine to set line dashes, odd and even dashes must have equal length. The SET DASHES routine enables the client to define dashes of varying length. The SET DASHES routine has the following format:

```
XSetDashes(display, gc_id, dash_offset, dash_list,
      dash_list_len)
```

The **dash_list_len** argument specifies the length of the dash list.

Figure 4–11 Line Defined Using GC Routines


Line Defined with GC Convenience Routines
Click MB1 to draw a dashed line.
Click MB3 to exit.

ZK–0102A–GE

4.4 Copying, Changing, and Freeing Graphics Contexts

In addition to defining a graphics context, clients can copy defined characteristics from one GC data structure into another. To copy a GC data structure, use COPY GC. The COPY GC routine has the following format:

```
XCopyGC(display, src_gc_id, gc_mask, dst_gc_id)
```

The **gc_mask** argument selects values to be copied from the source graphics context (**src_gc_id**). Use the method described in Section 4.2 for assigning values to a GRAPHICS CONTEXT.

The **dst_gc_id** argument specifies the new graphics context into which the server copies values.

After creating a graphics context structure, change values as needed using CHANGE GC. The following code fragment, which alters the values of the line drawn by Example 4–1, illustrates changing a graphics context structure:

```
    .
    .
    .
     xgcv.line_width = 10;
     xgcv.line_style = LineSolid;

     XChangeGC(dpy,gc,GCLineWidth | GCLineStyle,&xgcv);
    .
    .
    .
```

The example illustrates defining a new line style and width, and changing the graphics context to reflect the new values.

4.5 Using Graphics Characteristics Efficiently

The server must revalidate a graphics context whenever a client redefines it. Causing the server to revalidate a graphics context unnecessarily can seriously degrade performance.

The server revalidates a graphics context when one of the following conditions occurs:

- A client associates the graphics context with a different window.
- The graphics context clip list changes. Changes in the clip list can happen either when a client changes the graphics context clip origin or when the server modifies the clip list in response to overlapping windows.
- Any member of the graphics context changes.

To minimize revalidating the graphics context, submit as a group the requests to the server that identify the same window and graphics context. Grouping requests enables the server to revalidate the graphics context once instead of many times.

When it is necessary to change the value of graphics context members frequently, creating a new graphics context is more efficient than redefining an existing one, provided the client creates no more than 50 graphics contexts.

5 Using Color

Color is one attribute clients can define when creating a window or a graphics object. Depending on display hardware, clients can define color as black or white, as shades of gray, or as a spectrum of hues. Section 5.2 describes color definition in detail, including workstation types and the colors they support.

Xlib offers clients the choice of either sharing colors with other clients or allocating colors for exclusive use.

A client that does not have to change colors can share them with other clients. By sharing colors, the client saves color resources.

A client must allocate colors for its exclusive use when it needs to change them. For example, when presenting a graphic representation of a pipeline, the client might indicate flow through the pipeline by changing colors rather than redrawing the entire pipeline schematic. In this case, the client would allocate for exclusive use colors that represent pipeline flow.

This chapter introduces color management using Xlib and describes how to share and allocate color resources. The chapter includes the following topics:

- Color fundamentals—A description of pixels and planes, and color indices, cells, and maps
- Matching color requirements to screen types— How screen types affect color presentation
- Sharing color resources—How to share color resources with other clients
- Allocating colors for exclusive use—How to reserve colors for a single client
- Querying color resources—How to return values of color map entries
- Freeing color resources—How to release color resources

The concepts presented in this chapter apply to managing the color of both windows and graphic objects. Chapter 6 describes how to create graphic objects.

5.1 Pixels and Color Maps

The color of a window or graphics object depends on the values of pixels that constitute it. The number of bits associated with each pixel determines the number of possible pixel values. On a monochrome screen, one bit maps to each pixel. The number of possible pixel values is two. Pixels are either zero or one, black or white.

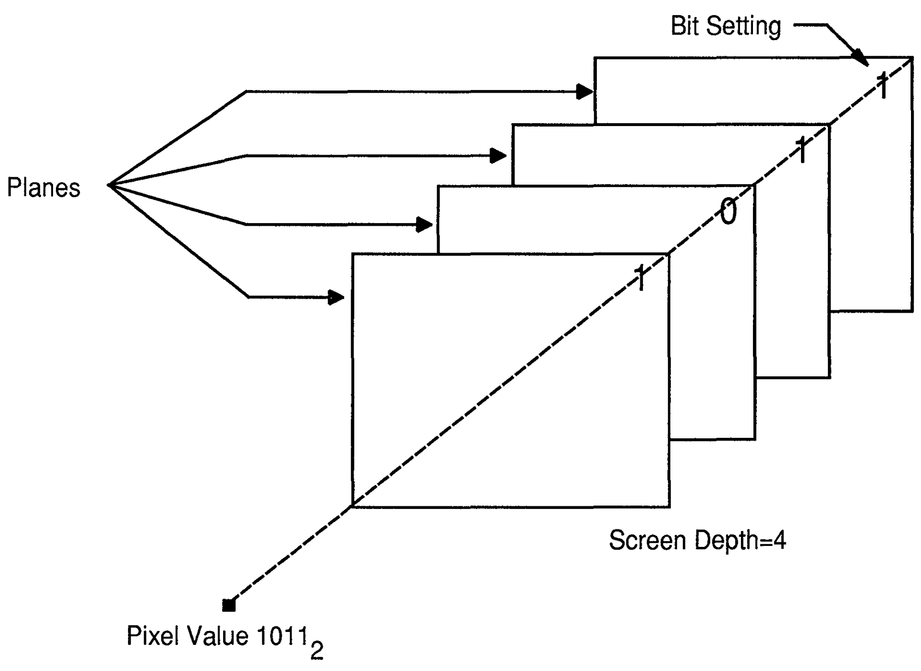
On a monochrome screen, all bits that define an image reside on one **plane**, an allocation of memory in which there is a one-to-one correspondence between bits and pixels. The number of planes is the **depth** of the screen.

The depth of intensity or color screens is greater than one. More than one bit defines the value of a pixel. Each bit associated with the pixel resides on a different plane.

The number of possible pixel values increases as depth increases. For example, if the screen has a depth of four planes (hardware will support a four-plane screen), the value of each pixel comprises four bits. Clients using a four-plane intensity display can produce up to sixteen levels of brightness. Clients using a four-plane color display can produce as many as sixteen colors.

Figure 5–1 illustrates the relationship between pixel values and planes.

Figure 5–1 Pixel Values and Planes

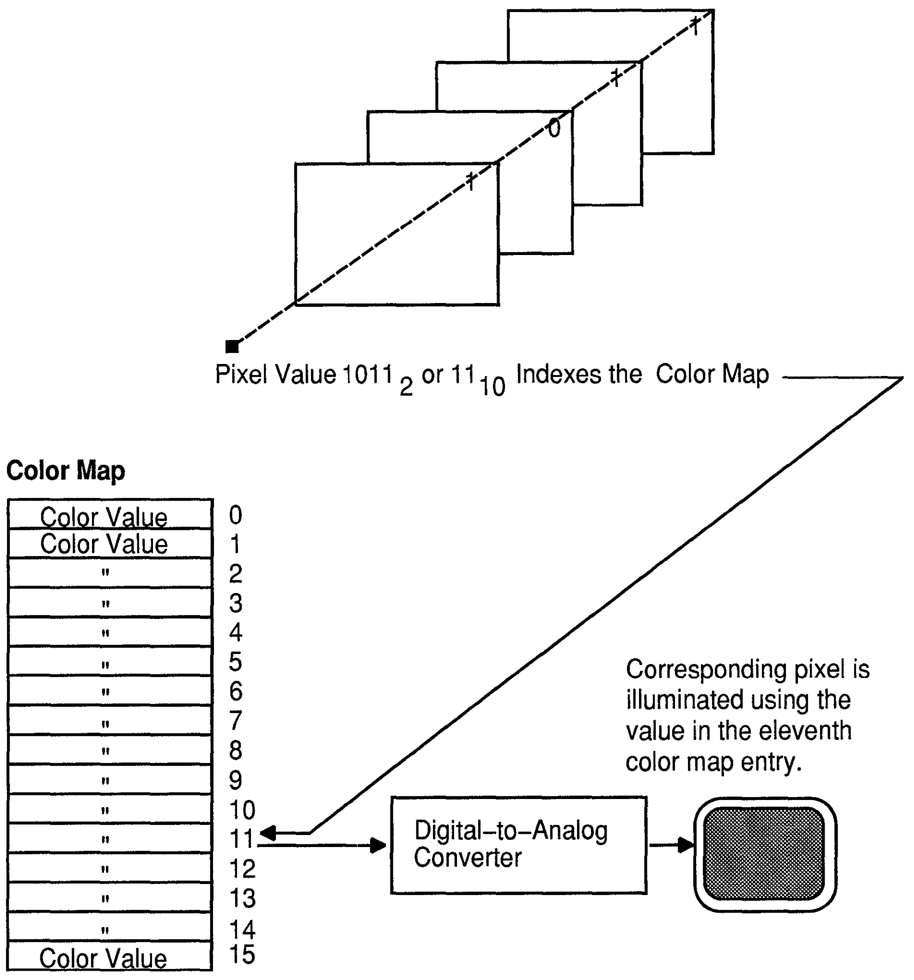


ZK–0074A–GE

Xlib uses **color maps** to define the color of each pixel. A color map contains a collection of **color cells**, each of which defines the color pixel value in terms of its red, green, and blue (RGB) components. Red, green, and blue components are in the range of zero (off) to 65535 (brightest) inclusive.

Each pixel value refers to a location in a color map, or is an **index** into a color map. For example, the pixel value illustrated in Figure 5–1 indexes color cell 11 in Figure 5–2.

Figure 5–2 Color Map, Cell, and Index


1
1
0
1
Pixel Value 1011 2 or 11 10 Indexes the Color Map
Color Map
Color Value
Color Value
"
Color Value
0
1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
Corresponding pixel is illuminated using the value in the eleventh color map entry.
Digital-to-Analog Converter

ZK-0076A-GE

Because most VAXstations have a hardware color map that is global to the entire display, clients should use the same color map whenever possible. Otherwise, some clients will appear in the wrong color.

For example, an image processing program that requires 128 colors might allocate and store a color map of these values. To alter some colors, another client may invoke a color palette program that chooses and mixes colors. The color palette program itself requires a color map, which the program allocates and installs.

Since both programs have allocated different color maps, this can produce undesirable results. When the image processing program runs, the color palette image may be incorrectly displayed because only the image processing color map is installed. Conversely, when the color palette program runs, the image processing program may be incorrectly displayed because only the color palette color map is installed.

Xlib reduces the problem of contending for color resources in two ways. First, Xlib provides a default color map to which all clients have access. Second, clients can either allocate color cells for exclusive use or allocate colors for shared use from the default color map. By sharing colors, a client can use the same color cells as other clients. This method conserves space in the default color map.

In cases where the client cannot use the default color map and must use a new color map, Xlib creates virtual color maps. The use of virtual color maps is analogous to the use of virtual memory in a multiprogramming environment where many processes must access physical memory. When concurrent processes collectively require more color map entries than exist in the hardware color map, the color values are swapped in and out of the hardware color map. However, swapping virtual color maps in and out of the hardware color map causes contention for color resources. Therefore, the client should avoid creating color maps whenever possible.

The process of loading or unloading color values of the virtual color map into the hardware lookup table occurs when a client calls the INSTALL COLORMAP or UNINSTALL COLORMAP routines. Typically, the privilege to install or remove color maps is restricted to the window manager.

5.2 Matching Color Requirements to Screen Types

Each screen has a list of **visual types** associated with it. The visual type identifies the characteristics of the screen, such as color or monochrome capability. Visual types partially determine the appearance of color on the screen and determine how a client can manipulate color maps for a specified screen.

Color maps can be manipulated in a variety of ways on some hardware, in a limited way on other hardware, and not at all on yet other hardware. For example, a screen may be able to display a full range of colors or a range of grays only, depending on its visual type.

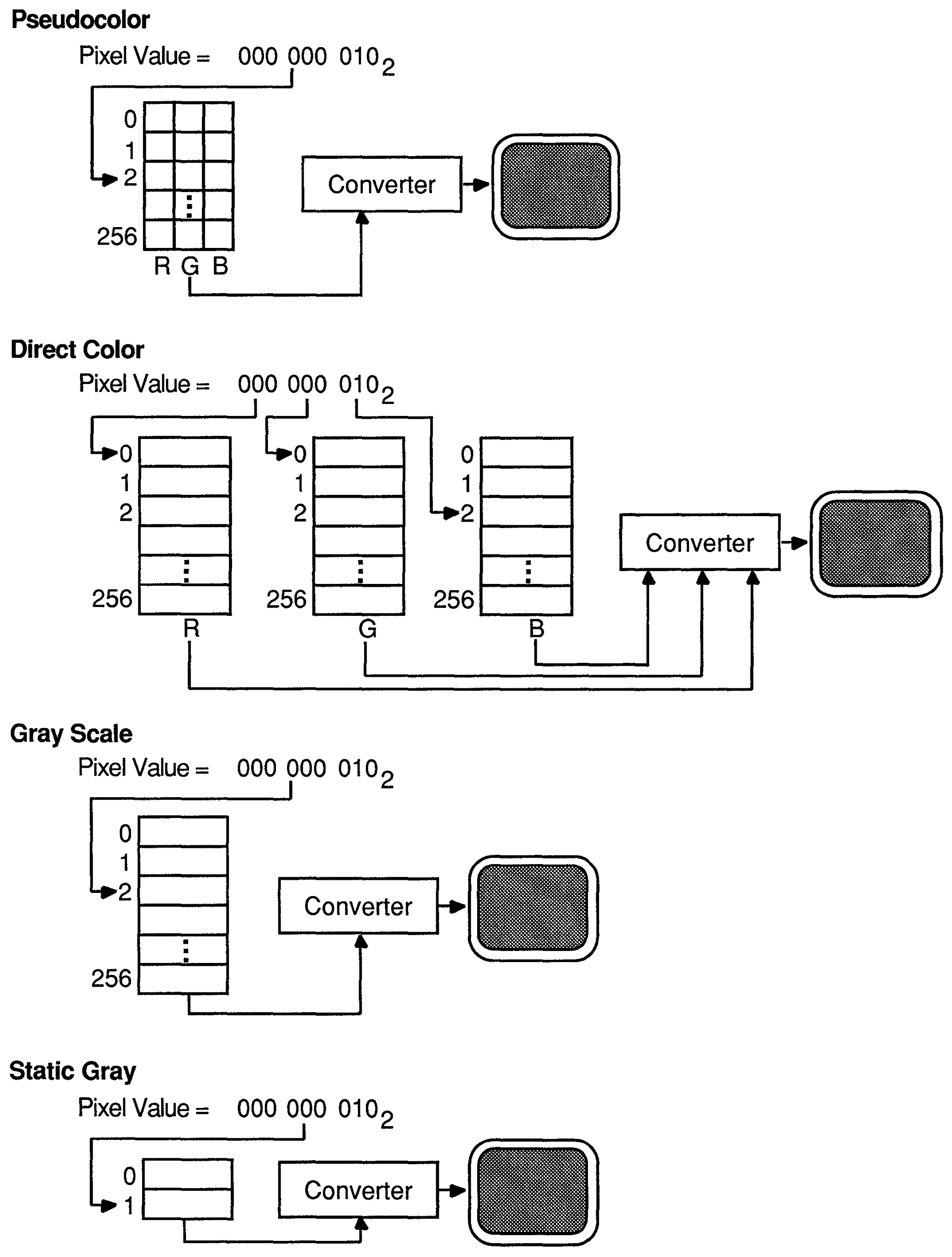
VMS DECwindows supports the following visual types:

- Pseudocolor—A pixel value indexes a color map to produce independent RGB values. RGB values can be changed dynamically, if a pixel has been allocated for exclusive use.
- Gray scale—Same as pseudocolor, except the pixel value indexes a color map that produces only shades of gray.
- Static gray—Same as gray scale, except that clients cannot change values in the color map.

In addition to supporting pseudocolor, gray scale, and static gray, VMS DECwindows enables clients to simulate the **direct color** visual type. Direct color stores RGB components into three separate data structures: one for red values, one for green values, and one for blue values. Pixel values refer to these three data structures, as Figure 5–3 illustrates. A direct color pixel value of 000000010, or 000 000 010, refers to member 0 of the data structure of red values, member 0 of the data structure of green values, and member 2 of the data structure of blue values.

See Section 5.4.2 for information about simulating a direct color device.

Figure 5–3 Visual Types and Color Map Characteristics



Default visual types are defined for each screen of a display and depend on the workstation and monitor type.

Table 5–1 lists VAXstations and their visual types.

Table 5–1 VAXstation Visual Types

	Visual Type	
VAXstation Type	**Monochrome Monitor**	**Color Monitor**
VAXstation II	Static gray	N/A
VAXstation 2000	Static gray	N/A
VAXstation II/GPX	Gray scale	Pseudocolor
VAXstation 2000/GPX	Gray scale	Pseudocolor
VAXstation 3200	Gray scale	Pseudocolor
VAXstation 3500	Gray scale	Pseudocolor

Before defining colors, use the following method to determine the visual type of a screen:

1 Use the DEFAULT VISUAL OF SCREEN routine to determine the identifier of the visual. Xlib returns the identifier to a visual data structure.

2 Refer to the class member of the data structure to determine the visual type.

The following example illustrates how to determine the visual type of a screen:

```
.
.
.
if ((XDefaultVisualOfScreen(screen))->class == PseudoColor
    ||  (XDefaultVisualOfScreen(screen))->class ==
        DirectColor)
.
.
.
```

5.3 Sharing Color Resources

Xlib provides the following ways to share color resources:

- Using named VMS DECwindows colors
- Specifying exact color values

The choice of using a named color or specifying an exact color depends on the needs of the client. For instance, if the client is producing a bar graph, specifying the named VMS DECwindows color “Red” as a color value may be sufficient, regardless of the hue that VMS DECwindows names “Red”. However, if the client is reproducing a portrait, specifying an exact red color value might be necessary to produce accurate skin tones.

Note that because of differences in hardware, no two monitors display colors exactly the same even though the same named colors are specified.

For a list of named VMS DECwindows colors, see Appendix C.

5.3.1 Using Named VMS DECwindows Colors

VMS DECwindows includes named colors that clients can share. To use a named color, call the ALLOC NAMED COLOR routine. ALLOC NAMED COLOR determines whether the color map defines a value for the specified color. If the color exists, the server returns the index to the color map. If the color does not exist, the server returns an error.

Example 5–1 illustrates specifying a color using ALLOC NAMED COLOR.

Example 5–1 Using Named VMS DECwindows Colors

```
static int doDefineColor(n)
{
    int pixel;
❶   XColor exact_color,screen_color;
❷   char *colors[ ] = {
        "dark slate blue",
        "light grey",
        "firebrick"
        };

    if ((XDefaultVisualOfScreen(screen))->class == PseudoColor
        || (XDefaultVisualOfScreen(screen))->class == DirectColor)
           {
❸       if (XAllocNamedColor(dpy, DefaultColormapOfScreen(screen),
            colors[n-1], &screen_color, &exact_color))
                return screen_color.pixel;
            else
                printf("Color not allocated!");

           }
    else
        printf("Not a color device!");
   .
   .
   .
```

❶ The client allocates storage for two color data structures. One, *exact_color*, defines the RGB values specified by the VMS DECwindows named color. The other, *screen_color*, defines the closest RGB values supported by the hardware.

For an illustration of the color data structure, see Section 5.3.2.

❷ An array of characters stores the names of the predefined VMS DECwindows colors that the client uses.

❸ The ALLOC NAMED COLOR routine has the following format:

```
XAllocNamedColor(display, colormap_id, color_name,
              screen_def_return, exact_def_return)
```

The client passes the names of VMS DECwindows colors by referring to the array *colors*.

5.3.2 Specifying Exact Color Values

To specify exact color values, use the following method:

1 Assign values to a color data structure

2 Call the ALLOC COLOR routine, specifying the color map that stores the definition. ALLOC COLOR returns a pixel value and changes the RGB values to indicate the closest color supported by the hardware.

Xlib provides a color data structure to enable clients to specify exact color values when sharing colors. (Routines that allocate colors for exclusive use and that query available colors also use the color data structure. For information about using the color data structure for these purposes, see Section 5.4.)

The following illustrates the color data structure:

```
typedef struct {
        unsigned long pixel;
        unsigned short red, green, blue;
        char flags;
        char pad;
} XColor;
```

Table 5–2 describes the members of the color data structure.

Table 5–2 Color Data Structure Members

Member Name	Contents	
pixel	Pixel value	
red	Specifies the red value of the pixel [1]	
green	Specifies the green value of the pixel [1]	
blue	Specifies the blue value of the pixel [1]	
flags	Defines which color components are to be changed in the color map. Possible flags are as follows:	
	DoRed	Sets red values
	DoGreen	Sets green values
	DoBlue	Sets blue values
pad	Makes the data structure an even length	

[1]Color values are scaled between 0 and 65535. "On full" in a color is a value of 65535, independent of the number of planes of the display. Half brightness in a color is a value of 32767; off is a value of 0. This representation gives uniform results for color values across displays with different color resolution.

Example 5–2 illustrates how to specify exact color definitions.

Example 5–2 Specifying Exact Color Values

```
/******* Create color ***********************/
static int doDefineColor(n)
{
    int pixel;
    XColor colors[3];

     if ((XDefaultVisualOfScreen(screen))->class == PseudoColor
     ||  (XDefaultVisualOfScreen(screen))->class == DirectColor)
        switch (n){
                 case 1:{
❶                    colors[n - 1].flags = DoRed | DoGreen | DoBlue;
❷                    colors[n - 1].red = 59904;
                     colors[n - 1].green = 44288;
                     colors[n - 1].blue = 59904;
❸                    if (XAllocColor(dpy, XDefaultColormapOfScreen(screen),
                          &colors[n - 1]))
                               return colors[n - 1].pixel;
                          else
                               printf("Color not allocated!");
                     return;
                     }
                 case 2:{
                     colors[n - 1].flags = DoRed | DoGreen | DoBlue;
                     colors[n - 1].red = 65280;
                     colors[n - 1].green = 0;
                     colors[n - 1].blue = 32512;
                     if (XAllocColor(dpy, XDefaultColormapOfScreen(screen),
                          &colors[n - 1]))
                               return colors[n - 1].pixel;
                          else
                               printf("Color not allocated!");
                     return;
                     }
                 case 3:{
                     colors[n - 1].flags = DoRed | DoGreen | DoBlue;
                     colors[n - 1].red = 37632;
                     colors[n - 1].green = 56064;
                     colors[n - 1].blue = 28672;
                     if (XAllocColor(dpy, XDefaultColormapOfScreen(screen),
                         &colors[n - 1]))
                               return colors[n - 1].pixel;
                          else
                               printf("Color not allocated!");
                     return;
                     }
        }
     else
        switch (n) {
            case 1:                 return XBlackPixelOfScreen(screen); break;
            case 2:                 return XWhitePixelOfScreen(screen); break;
            case 3:                 return XBlackPixelOfScreen(screen); break;
        }
}
```

❶ Specify that RGB values are defined.

❷ Define color values in the first of three color data structures.

❸ After defining RGB values, call the ALLOC COLOR routine. ALLOC COLOR allocates shared color cells on the default color map and returns a pixel value for the color that matches the specified color most closely.

5.4 Allocating Colors for Exclusive Use

A client that does not need to change color values should share colors using the methods described in Section 5.3.2. Sharing colors saves resources. However, a client that changes color values must allocate them for its exclusive use.

Xlib provides two methods for allocating colors for the exclusive use of a client. First, the client can allocate cells and store color values in the default color map. Second, if the default color map does not contain enough storage, the client can create its own color map and store color values in it.

This section describes how to specify a color map, how to allocate cells for exclusive use, and how to store values in the color cells.

5.4.1 Specifying a Color Map

Clients can either use the default color map and allocate its color cells for exclusive use or create their own color maps.

If possible, use the default color map. Although a client can create color maps for its own use, the hardware color map storage is limited. When a client creates its own color map, the map must be loaded, or installed, into the hardware color map before the client map can be used. If the client color map is not installed, the client may refer to a different color map and possibly display the wrong color. Using the default color map eliminates this problem. See Section 5.1 for information about how Xlib handles color maps.

To specify the default color map, use the DEFAULT COLORMAP routine. DEFAULT COLORMAP returns the identifier of the default color map.

If the default color map does not contain enough resources, the client can create its own color map.

To create a color map, use the following method:

1 Determine the visual type of a specified screen using the method described in Section 5.2

2 Call the CREATE COLORMAP routine.

The CREATE COLORMAP routine creates a color map for the specified window and visual type. CREATE COLORMAP has the following format:

```
XCreateColormap(display, window_id, visual_struc, alloc)
```

The **alloc** argument specifies whether the client creating the color map allocates all of the color map entries for its exclusive use or creates a color map with no defined color map entries. To allocate all entries for exclusive use, specify the constant **AllocAll**. To allocate no defined map entries, specify the constant **AllocNone**. The latter is useful when two or more clients are to share the newly created color map.

If the visual type is pseudocolor or gray scale, the client can either allocate all or no map entries. If the visual type is static gray, the client must allocate no entries.

See Section 5.4.2 for information about allocating colors. See Example 5–3 for an example of specifying the default color map.

5.4.2 Allocating Color Cells

After specifying a color map, allocate color cells in it.

To allocate color cells, call the ALLOC COLOR CELLS routine to allocate cells for a pseudocolor device or a gray scale device. Call the ALLOC COLOR PLANES routine to simulate a direct color device. See Section 5.2 for information about the direct color visual type.

Example 5–3 illustrates how to allocate colors for exclusive use. The program creates a color wheel that rotates when the user presses MB1.

Example 5–3 Allocating Colors for Exclusive Use

```
#include <decw$include/Xlib.h>
#include <decw$include/Xutil.h>
#include math;

#define winW 600
#define winH 600
#define backW 800
#define backH 800

Display *dpy;
Window win;
Pixmap pixmap;
Colormap map;
GC gc;
Screen *screen;
XColor *colors;
int offsetX, offsetY;
int fullcount;
int ButtonIsDown = 0;
int n, exposeflag = 0;
int ihop=1;
XSetWindowAttributes xswa;
```

(continued on next page)

Example 5–3 (Cont.) Allocating Colors for Exclusive Use

```
static void doInitialize( );
static void doCreateWindows( );
static void doCreateGraphicsContext( );
static void doCreatePixmap( );
static void doCreateColor( );
static void doCreateWheel( );
static void doWMHints( );
static void doMapWindows( );
static void doHandleEvents( );
static void doExpose( );
static void doButtonPress( );
static void doButtonRelease( );
static void doChangeColors( );
static void doLoadColormap( );
static void doHLS_to_RGB( );
static void doConfigure( );

/********************** The main program *******************************/

static int main()
{
    doInitialize( );
    doHandleEvents( );
}

/***************** doInitialize *************************/
static void doInitialize( )
{
    dpy = XOpenDisplay(0);

    screen = DefaultScreenOfDisplay(dpy);

    doCreateWindows( );

    doCreateGraphicsContext( );

    doCreatePixmap( );

    doCreateColor( );

    doCreateWheel( );

    doWMHints( );

    doMapWindows( );
}
/******* doCreateWindows *********/
static void doCreateWindows( )
{
    int winX = 100;
    int winY = 100;

    /* Create the win window */

    xswa.event_mask = ExposureMask | ButtonPressMask |
        ButtonReleaseMask | StructureNotifyMask;
    xswa.background_pixel = XBlackPixelOfScreen(screen);

    win = XCreateWindow(dpy, RootWindowOfScreen(screen),
        winX, winY, winW, winH, 0,
        DefaultDepthOfScreen(screen), InputOutput,
        DefaultVisualOfScreen(screen), CWEventMask | CWBackPixel, &xswa);
}
```

(continued on next page)

Example 5–3 (Cont.) Allocating Colors for Exclusive Use

```
/******** Create the graphics context *********/
static void doCreateGraphicsContext( )
{
    XGCValues xgcv;

    /* Create graphics context. */

    gc = XCreateGC(dpy, win, 0, 0);
    XSetForeground(dpy, gc, XWhitePixelOfScreen(screen));
}

/******* doCreatePixmap *********/
static void doCreatePixmap( )
{
❶   pixmap = XCreatePixmap(dpy, XRootWindow(dpy, XDefaultScreen(dpy)),
        backW, backH, XDefaultDepthOfScreen(screen));
    XFillRectangle(dpy, pixmap, gc, 0, 0, backW, backH);

}
/******* doCreateColor ********/
❷static void doCreateColor( )
{
int *pixels;
int contig;
int *plane_masks;

        if ((XDefaultVisualOfScreen(screen))->class != PseudoColor &&
            (XDefaultVisualOfScreen(screen))->class != DirectColor)
            {
                sys$exit(1);
            }

❸      map = XDefaultColormapOfScreen(screen);

        xswa.colormap = map;
        fullcount = XDisplayCells(dpy, XDefaultScreen(dpy))/2;
        if (fullcount > 128) fullcount = 128;
        pixels = malloc(sizeof(int)*fullcount);
        if (!XAllocColorCells(dpy, map, contig, plane_masks,
            0, pixels, fullcount))
            {
                sys$exit(1);
            }
        colors = malloc(sizeof(XColor)*fullcount);
        doLoadColormap(pixels);
}
/***** doCreateWheel *****/
❹static void doCreateWheel( )
{
int pixel, i, j;
XPoint *pgon;
int xcent, ycent;

       pixel = XWhitePixelOfScreen(screen);
```

(continued on next page)

Example 5–3 (Cont.) Allocating Colors for Exclusive Use

```
    /* Now set up wheel.  It is really a set of triangles*/
        pgon = malloc(sizeof(XPoint)*3*fullcount+1);
        xcent=backW/2;
        ycent=backH/2;
❺       pgon[0].x = backW;
        pgon[0].y = backH/2;

    /* Fill in coordinate for center point in all triangles */

        for (i=0;i<fullcount*3;i+=3)
            {
            pgon[i+1].x = xcent;
            pgon[i+1].y = ycent;
            }

     /* Calculate the triangle points on the outer circle */

        for (pixel=0,i=0;pixel<fullcount;i+=3, pixel++)
            {
            double x,y,xcent_f,ycent_f;
            xcent_f = (double)xcent;
            ycent_f = (double)ycent;
            x=cos( (((double)pixel+1.)/(double)fullcount)*2.*3.14159);
            y=sin( (((double)pixel+1.)/(double)fullcount)*2.*3.14159);
            pgon[i+2].x = (int)(x*xcent_f)+xcent;
            pgon[i+2].y = (int)(y*ycent_f)+ycent;
            pgon[i+3].x = pgon[i+2].x;
            pgon[i+3].y = pgon[i+2].y;
            XSetForeground(dpy, gc, colors[i/3].pixel);
            XFillPolygon(dpy, pixmap, gc, &pgon[i], 3, Convex, CoordModeOrigin);
            }
    offsetX = (backW - winW)/2;
    offsetY = (backH - winH)/2;
    return;
}
/******** do WMHints *************/
static void doWMHints( )
{
    XSizeHints xsh;

    /* Define the size and name of the win window */

    xsh.x = 100;
    xsh.y = 100;
    xsh.width = winW;
    xsh.height = winH;
    xsh.flags = PPosition | PSize;

    XSetNormalHints(dpy, win, &xsh);

    XStoreName(dpy, win, "Color Wheel: Press MB1 to Rotate or Click MB2 to Exit.");
}


/******** doMapWindows ***********/
static void doMapWindows( )
{
    XMapWindow(dpy, win);
}
```

(continued on next page)

Example 5–3 (Cont.) Allocating Colors for Exclusive Use

```
/***************** doHandleEvents **********************/
static void doHandleEvents( )
{
    XEvent event;

    for ( ; ; ) {
        XNextEvent(dpy, &event);

        switch (event.type) {
            case Expose:                doExpose(&event); break;
            case ButtonPress:           doButtonPress(&event); break;
            case ButtonRelease:         doButtonRelease(&event); break;
            case ConfigureNotify:       doConfigure(&event); break;
        }
    }
}


/***** Handle window exposures *****/
static void doExpose(eventP)
XEvent *eventP;
{
❻   XCopyArea(dpy, pixmap, win, gc, offsetX + eventP->xexpose.x,
        offsetY + eventP->xexpose.y, eventP->xexpose.width,
        eventP->xexpose.height, eventP->xexpose.x, eventP->xexpose.y);

}


/******** doButtonPress ************/
static void doButtonPress(eventP)
XEvent *eventP;
{
    if (eventP ->xbutton.button == Button2) {
        sys$exit (1);
    }
    ButtonIsDown = 1;
    if (ButtonIsDown) doChangeColors( );
    return;
}

/******** doButtonRelease ************/
static void doButtonRelease(eventP)
XEvent *eventP;
{
    ButtonIsDown = 0;
    return;
}


/*********doConfigure *******************/
static void doConfigure(eventP)
XEvent *eventP;
{
❼   offsetX = (backW - eventP->xconfigure.width)/2;
    offsetY = (backH - eventP->xconfigure.height)/2;
}
```

(continued on next page)

Example 5–3 (Cont.) Allocating Colors for Exclusive Use

```
/*************doChangeColors************/
❽static void doChangeColors( )
{
      for (;!(XPending(dpy));){
                unsigned int i,temp;
                double h,r,g,b;

                temp = colors[0].pixel;
                for (i=0;i<fullcount-1;i++)
                      colors[i].pixel = colors[i+1].pixel;
                colors[fullcount-1].pixel = temp;
                XStoreColors(dpy, map, colors, fullcount);
      }
}

/**** doLoadColormap ****/
❾static void doLoadColormap(pPixels)
int *pPixels;
{

    unsigned int i,j;
    double h,r,g,b;

        for (i=0;i < fullcount;i++) {
            colors[i].pixel=pPixels[i];
            colors[i].flags = DoRed | DoGreen |DoBlue;
        }
        for (i=0; i < fullcount ; i++) {
❿            h = (double)i*360/((double)fullcount+1);
            doHLS_to_RGB(&h,&.5,&.5,&r,&g,&b);
            colors[i].red = r * 65535.0;
            colors[i].green = g * 65535.0;
            colors[i].blue = b * 65535.0;
        }
      XStoreColors(dpy, map, colors, fullcount);
}


/**** doHLS_to_RGB ****/

static void doHLS_to_RGB (h,l,s, r,g,b)
double *h,*l,*s,*r,*g,*b;
{
     double m1,m2;
     double value();

     m2 = (*l < 0.5) ? (*l)*(1+*s) : *l + *s- (*l)*(*s) ;
     m1 = 2*(*l) - m2;
        if ( *s == 0 )
          { (*r)=(*g)=(*b)=(*l); }
        else
          { *r=value(m1,m2,(double)(*h+120.));
            *g=value(m1,m2,(double)(*h+000.));
            *b=value(m1,m2,(double)(*h-120.));
          }
        return;
}
```

(continued on next page)

Example 5–3 (Cont.) Allocating Colors for Exclusive Use

```
double value (n1,n2,hue)
double n1,n2,hue;
{
        double val;

        if (hue>360.)  hue -= 360.;
        if (hue<0.  )  hue += 360.;

        if (hue<60)
          val = n1+(n2-n1)*hue/60.;
        else if (hue<180.)
          val = n2;
        else if (hue<240.)
          val = n1+(n2-n1)*(240.-hue)/60.;
        else
          val = n1;
        return (val);
 }
```

❶ The client uses a pixmap as a backing store for the color wheel. When a user reconfigures the color wheel window, the client copies the color wheel from the pixmap into the resized window. For information about creating and using pixmaps, see Chapter 7.

❷ After creating the pixmap for backing store, the client creates colors for the wheel and the wheel itself. The client-defined *doCreateColor* routine allocates color cells for the exclusive use of the client and stores initial color values in the color map.

❸ The client uses the default color map, specifying that only 128 color cells be allocated. After allocating color cells, the client calls the client-defined *doLoadColormap* routine to define color values. For a description of the routine, see callouts 7, 8, 9, and 10.

❹ The client-defined *doCreateWheel* routine defines the wheel used to display colors and specifies initial color values.

❺ The wheel is composed of polygons. Each polygon is defined by three points, one in the center of the wheel and two at the circumference. After the initial polygon is specified, each polygon shares one point with the polygon previously defined, as Figure 5–4 illustrates.

To define each point the client uses a point data structure, which is described in Chapter 6. After defining a polygon, the client fills it with a specified foreground color.

❻ When the user reconfigures the window, the server generates an expose event. In response to the event, the client copies the pixmap into the exposed area, which is calculated using the offset from the original to the new position of the window. For information about handling exposure events, see Chapter 9.

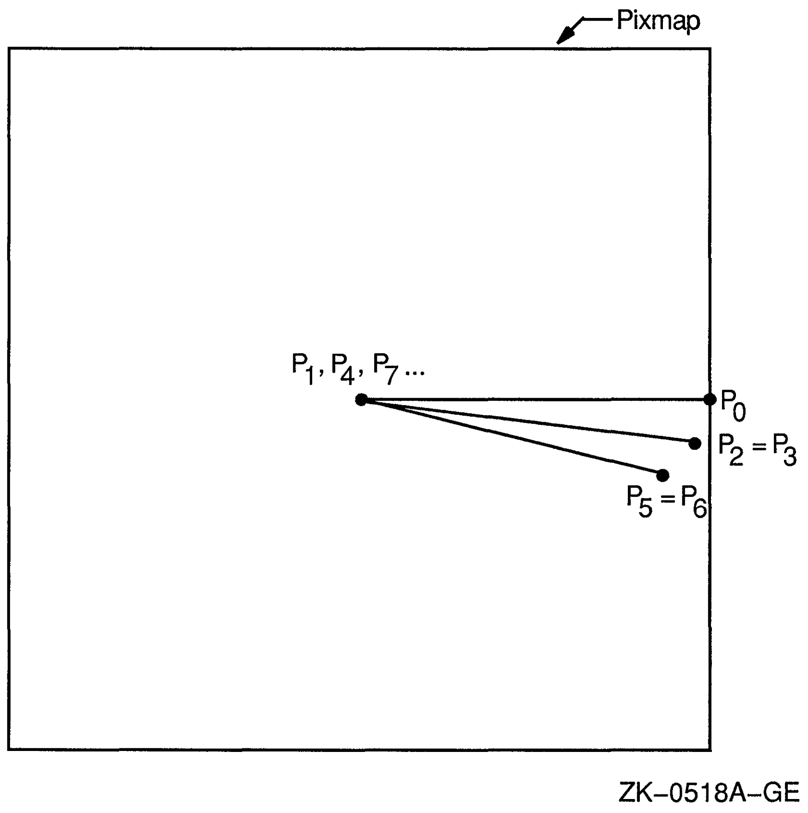
❼ The client calculates the offset from the original window position in response to a configure notify event. The server issues a configure notify event each time the user resizes the color wheel window. For information about handling configure notify events, see Chapter 9.

8 The rotation of the color wheel is accomplished by changing values in the color map. As long as there are no pending events, and the user is pressing MB1, the client-defined *doChangeColors* routine shifts color values by one.

9 The *doLoadColormap* routine initializes the color wheel by defining 128 colors and storing them in the color map.

10 Colors are defined initially using the Hue, Light, Saturation (HLS) system. The values of color hues vary, while values for light and saturation remain constant. After a color has been defined using HLS, the color is converted into RGB values by the client-defined *doHLS_to_RGB* routine. When all colors are defined, the client stores them in the color map by calling the STORE COLORS routine.

Figure 5–4 Polygons That Define the Color Wheel



When allocating colors from any shared color map, the client may exhaust the resources of the color map. In this case, Xlib provides a routine for copying the default color map entries into a new client-created color map.

To create a new color map when the client exhausts the resources of a previously shared color map, use the COPY COLORMAP AND FREE routine. The routine creates a color map of the same visual type and for the same screen as the previously shared color map. The previously shared color map can be either the default color map or a client-created color map. The COPY COLORMAP AND FREE routine has the following format:

```
XCopyColormapAndFree(display, colormap_id)
```

COPY COLORMAP AND FREE copies all allocated cells from the previously shared color map to the new color map, keeping color values intact. The new color map is created with the same value of the argument **alloc** as the previously shared color map and has the following effect on the new color map entries:

Value of alloc	Effect
AllocAll	All entries are copied from the previously shared color map and are then freed to create writable map entries.
AllocNone	The entries moved are all pixels and planes that have been allocated using the following routines and that have not been freed since they were allocated: ALLOC COLOR, ALLOC NAMED COLOR, ALLOC COLOR CELLS, ALLOC COLOR PLANES.

5.4.3 Storing Color Values

After allocating color entries in the color map, store RGB values in the color map cells using the following method:

1 Assign color values to the color data structure and set the flags member to indicate the values defined.

2 Call the STORE COLOR routine to store one color, the STORE COLORS routine to store more than one color, or the STORE NAMED COLOR routine to store a named color.

The STORE COLOR routine has the following format:

```
XStoreColor(display, colormap_id, screen_def_return)
```

The STORE COLORS routine has the following format:

```
XStoreColors(display, colormap_id, screen_defs_return,
      num_colors)
```

The STORE NAMED COLOR routine has the following format:

```
XStoreNamedColor(display, colormap_id, color_name,
      pixel, flags)
```

5.5 Freeing Color Resources

To free storage allocated for client colors, call the FREE COLORS routine. FREE COLORS releases all storage allocated by the following color routines: ALLOC COLOR, ALLOC COLOR CELLS, ALLOC NAMED COLORS, ALLOC COLOR PLANES.

To delete the association between the color map ID and the color map, use the FREE COLORMAP routine. FREE COLORMAP has no effect on the default color map of the screen. If the color map is an installed color map, FREE COLORMAP removes it.

5.6 Querying Color Map Entries

Xlib provides routines to return both the RGB values of the color map index and the name of a color.

To query the RGB values of a specified pixel in the color map, use the QUERY COLOR routine. The value returned is the value passed in the pixel member of the color data structure.

To query the RGB values of an array of pixel values, use the QUERY COLORS routine. The values returned are the values passed in the pixel member of the color data structure.

To look up the values associated with a named color, use the LOOKUP COLOR routine. LOOKUP COLOR uses the specified color map to find out the values with respect to a specific screen. It returns both the exact RGB values and the closest RGB values supported by hardware.

6 Drawing Graphics

Xlib provides clients with routines that draw graphics into windows and pixmaps. This chapter describes how to create and manage graphics drawn into windows, including the following topics:

- Drawing points, lines, rectangles, and arcs
- Filling rectangles, polygons, and arcs
- Copying graphics
- Limiting graphics to a region of a window or pixmap
- Clearing graphics from a window
- Creating cursors

Chapter 7 describes drawing graphics into pixmaps.

6.1 Graphics Coordinates

Xlib graphics coordinates define the position of graphics drawn in a window or pixmap. Coordinates are either relative to the origin of the window or pixmap in which the graphics object is drawn or relative to a previously drawn graphics object.

Xlib graphics coordinates are similar to the coordinates that define window position. Xlib measures length along the *x* axis from the origin to the right. Xlib measures length along the *y* axis from the origin down. Xlib specifies coordinates in units of pixels.

6.2 Using Graphics Routines Efficiently

If clients use the same drawable and graphics context for each call, Xlib handles back to back calls of DRAW POINT, DRAW LINE, DRAW SEGMENT, DRAW RECTANGLE, FILL ARC, and FILL RECTANGLE in a batch. Batching increases efficiency by reducing the number of requests to the server.

When drawing more than a single point, line, rectangle, or arc, clients can also increase efficiency by using routines that draw or fill multiple graphics (DRAW POINTS, DRAW LINES, DRAW SEGMENTS, DRAW RECTANGLES, DRAW ARCS, FILL ARCS, and FILL RECTANGLES). Clipping negatively affects efficiency. Consequently, clients should ensure that graphics they draw to a window or pixmap are within the boundary of the drawable. Drawing outside the window or pixmap decreases performance. Clients should also ensure that windows into which they are drawing graphics are not occluded.

The most efficient method for clearing multiple areas is using the FILL RECTANGLES routine. By using the FILL RECTANGLES routine, clients can increase server performance. For information about using FILL RECTANGLES to clear areas, see Section 6.6.1.

6.3 Drawing Points and Lines

Xlib includes routines that draw points and lines. When clients draw more than one point or line, performance is most efficient if they use Xlib routines that draw multiple points or lines rather than calling single point and line-drawing routines many times.

This section describes using routines that draw both single and multiple points and lines.

6.3.1 Drawing Points

To draw a single point, use the DRAW POINT routine, specifying *x* and *y* coordinates, as in the following:

```
    .
    .
    .
    int x,y=100;
    XDrawPoint(display, window, gc, x, y);
```

If drawing more than one point, use the following method:

1 Define an array of point data structures.

2 Call the DRAW POINTS routine, specifying the array that defines the points, the number of points the server is to draw, and the coordinate system the server is to use. The server draws the points in the order specified by the array.

Xlib includes the point data structure to enable clients to define an array of points easily. The following illustrates the data structure:

```
typedef struct {
    short x, y;
} XPoint;
```

Table 6–1 describes the members of the point data structure.

Table 6–1 Point Data Structure Members

Member Name	Contents
x	Defines the x value of the coordinate of a point
y	Defines the y value of the coordinate of a point

The server determines the location of points according to the following:

- If the client specifies the constant **CoordModeOrigin**, the server defines all points in the array relative to the origin of the drawable.

- If the client specifies the constant **CoordModePrevious**, the server defines the coordinates of the first point in the array relative to the origin of the drawable and the coordinates of each subsequent point relative to the point preceding it in the array.

The server refers to the following members of the GC data structure to define the characteristics of points it draws:

Function	Plane mask
Foreground	Subwindow mode
Clip x origin	Clip y origin
Clip mask	

Chapter 4 describes GC data structure members.

Example 6–1 uses the DRAW POINTS routine to draw a circle of points each time the user clicks MB1.

Figure 6–1 illustrates sample output from the program.

Example 6–1 Drawing Multiple Points

```
    /* Create window win on                                *
     *  display dpy, defined as follows:                   *
     *     Position: x = 100,y = 100                       *
     *     Width = 600                                     *
     *     Height = 600                                    *
     *  gc refers to the graphics context                  */
    .
    .
    .
/****************** doHandleEvents **********************/
static void doHandleEvents( )
{
    XEvent event;

    for ( ; ; ) {
        XNextEvent(dpy, &event);
        switch (event.type) {
            case Expose:           doExpose(&event); break;
            case ButtonPress:      doButtonPress(&event); break;
        }
    }
}

/***** Write a message *****/
❶static void doExpose(eventP)
XEvent *eventP;
{
    char message1[ ] = {"To create points, click MB1"};
    char message2[ ] = {"Each click creates a new circle of points"};
    char message3[ ] = {"To exit, click MB2"};

    XDrawImageString(dpy, win, gc, 150, 25, message1, strlen(message1));
    XDrawImageString(dpy, win, gc, 150, 50, message2, strlen(message2));
    XDrawImageString(dpy, win, gc, 150, 75, message3, strlen(message3));
}
```

(continued on next page)

Example 6–1 (Cont.) Drawing Multiple Points

```
/***** Draw the points*****/
static void doButtonPress(eventP)
XEvent *eventP;
{
#define POINT_CNT 100
#define RADIUS 50
    XPoint point_arr[POINT_CNT];
    int i;
❷   int x = eventP->xbutton.x;
    int y = eventP->xbutton.y;

    if (eventP->xbutton.button == Button2) sys$exit (1);

    for (i=0;i<POINT_CNT;i++) {
        point_arr[i].x = x + RADIUS*cos(i);
        point_arr[i].y = y + RADIUS*sin(i);
    }

❸   XDrawPoints(dpy, win, gc, &point_arr, POINT_CNT, CoordModeOrigin);
}
```

❶ When the client receives notification that the server has mapped the window, the *doExpose* routine writes three messages into the window. For information about using the DRAW IMAGE STRING routine, see Chapter 8.

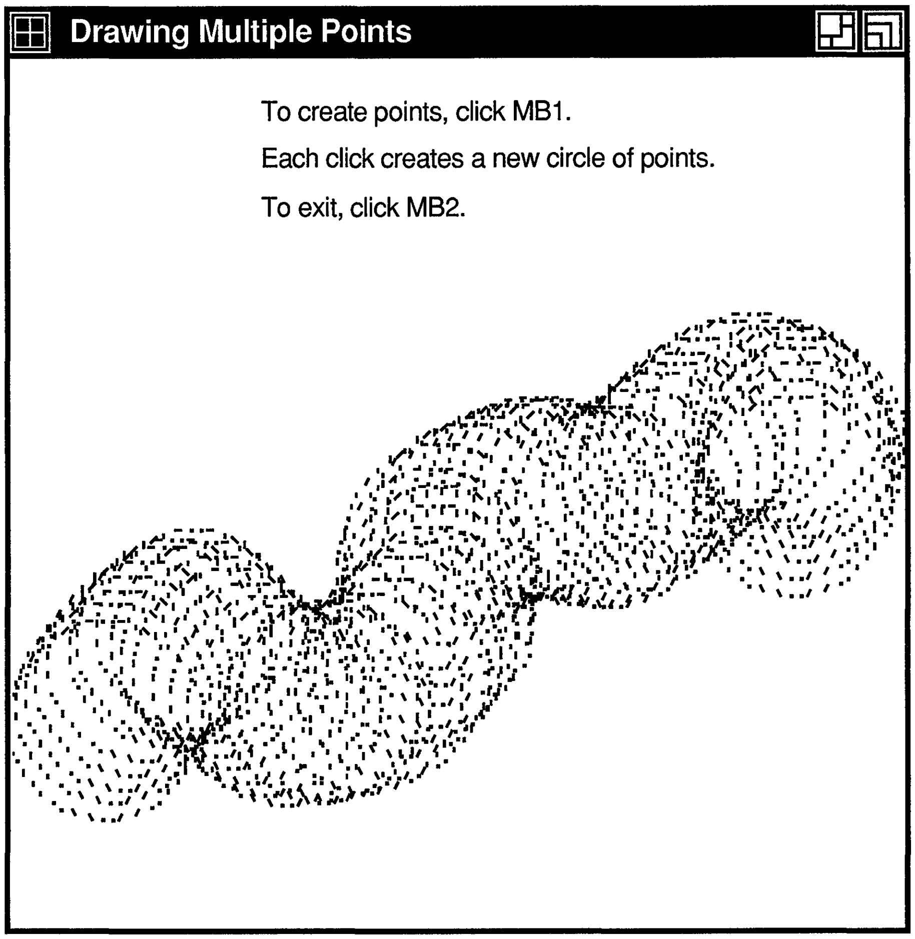
❷ If the user clicks any mouse button, the client initiates the *doButtonPress* routine. If the user clicks MB1, the client draws 50 points. If the user clicks MB2, the client exits the system. The client determines which button the user pressed by referring to the button member of the button event data structure. For more information about the button event data structure, see Chapter 9.

❸ The DRAW POINTS routine has the following format:

```
XDrawPoints(display, drawable_id, gc_id, points,
         num_points, point_mode)
```

The **point_mode** argument specifies whether coordinates are relative to the origin of the drawable or to the previous point in the array.

Figure 6–1 Circles of Points Created Using the DRAW POINTS Routine



To create points, click MB1.
Each click creates a new circle of points.
To exit, click MB2.

ZK-0107A-GE

6.3.2 Drawing Lines and Line Segments

Xlib includes routines that draw single lines, multiple lines, and line segments. To draw a single line, use the DRAW LINE routine, specifying beginning and ending points, as in the following:

```
 .
 .
 .
int x1,y1=100;
int x2,y2=200;
XDrawLine(display, window, gc, x1, y1, x2, y2);
```

To draw multiple lines, use the following method:

1 Define an array of points using the point data structure described in Section 6.3.1 to specify beginning and ending line points. The server interprets pairs of array elements as beginning and ending points. For

example, if the array that defines the beginning point is *point*[*i*], the server reads *point*[*i* + 1] as the corresponding ending point.

2 Call the DRAW LINES routine, specifying the following:

- The array that defines the points.
- The number of points that define the line.
- The coordinate system the server uses to locate the points. The server draws the lines in the order specified by the array.

Clients can specify either the **CoordModeOrigin** or the **CoordModePrevious** constant to indicate how the server determines the location of beginning and ending points. The server uses the methods described in Section 6.3.1.

The server draws lines in the order the client has defined them in the point data structure. Lines join correctly at all intermediate points. If the first and last points coincide, the first and last line also join correctly. For any given line, the server draws pixels only once. The server draws intersecting pixels multiple times if zero-width lines intersect; it draws intersecting pixels of wider lines only once.

Example 6–2 uses the DRAW LINES routine to draw a star when the server notifies the client that the window is mapped.

Example 6–2 Drawing Multiple Lines

```
    /* Create window win on                             *
     *  display dpy, defined as follows:                *
     *     Position: x = 100,y = 100                    *
     *     Width = 600                                  *
     *     Height = 600                                 *
     * gc refers to the graphics context                */
    .
    .
    .

/***************** doHandleEvents **********************/
static void doHandleEvents( )
{
    XEvent event;

    for ( ; ; ) {
        XNextEvent(dpy, &event);
        switch (event.type) {
            case Expose:            doExpose(&event); break;
        }
    }
}

/***** doExpose ****/
static void doExpose(eventP)
XEvent *eventP;
{
    XPoint pt_arr[6];
```

(continued on next page)

Example 6–2 (Cont.) Drawing Multiple Lines

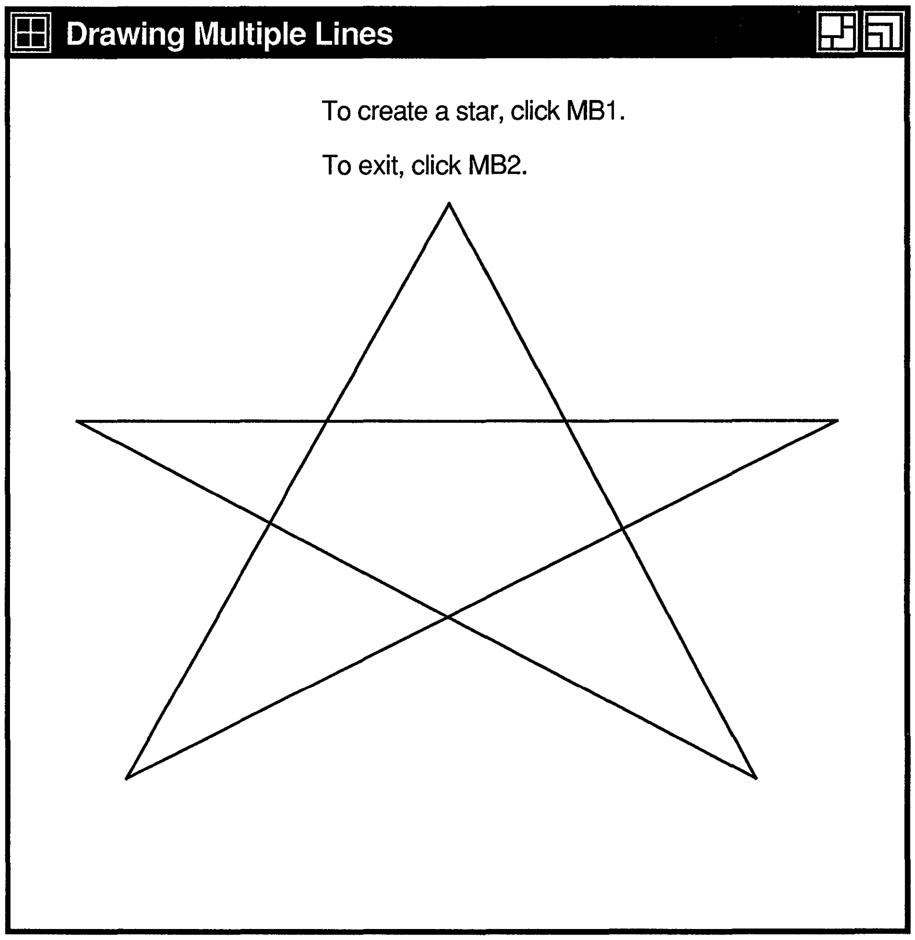
```
❶  pt_arr[0].x = 75;
   pt_arr[0].y = 500;
   pt_arr[1].x = 300;
   pt_arr[1].y = 100;
   pt_arr[2].x = 525;
   pt_arr[2].y = 500;
   pt_arr[3].x = 50;
   pt_arr[3].y = 225;
   pt_arr[4].x = 575;
   pt_arr[4].y = 225;
   pt_arr[5].x = 75;
   pt_arr[5].y = 500;

❷   XDrawLines(dpy, win, gc, &pt_arr, 6, CoordModeOrigin);
}
   .
   .
   .
```

❶ The *doExpose* routine uses point data structures to define beginning and ending points of lines.

❷ The call to draw lines refers to a graphics context (*gc*), which the client has previously defined, and an array of point data structures. The constant **CoordModeOrigin** indicates that all points are relative to the origin of *win* (100,100).

Figure 6–2 illustrates the resulting output.

Figure 6–2 Star Created Using the DRAW LINES Routine



ZK–0103A–GE

Use the DRAW SEGMENTS routine to draw multiple, unconnected lines, defining an array of segments in the segment data structure. The following illustrates the data structure:

```
typedef struct {
    short x1, y1, x2, y2;
} XSegment;
```

Table 6–2 describes the members of the data structure.

Table 6–2 Segment Data Structure Members

Member Name	Contents
x1	The x value of the coordinate that specifies one endpoint of the segment
y1	The y value of the coordinate that specifies one endpoint of the segment
x2	The x value of the coordinate that specifies the other endpoint of the segment
y2	The y value of the coordinate that specifies the other endpoint of the segment

DRAW SEGMENTS functions like the DRAW LINES routine, except the routine does not use the coordinate mode.

The DRAW LINE and DRAW SEGMENTS routines refer to all but the join style, fill rule, arc mode, and font members of the GC data structure to define the characteristics of lines. The DRAW LINES routine refers to all but the fill rule, arc mode, and font members of the data structure.

Chapter 4 describes the GC data structure.

6.4 Drawing Rectangles and Arcs

As with routines that draw points and lines, Xlib provides clients the choice of drawing either single or multiple rectangles and arcs. If a client is drawing more than one rectangle or arc, use the multiple-drawing routines for most efficiency.

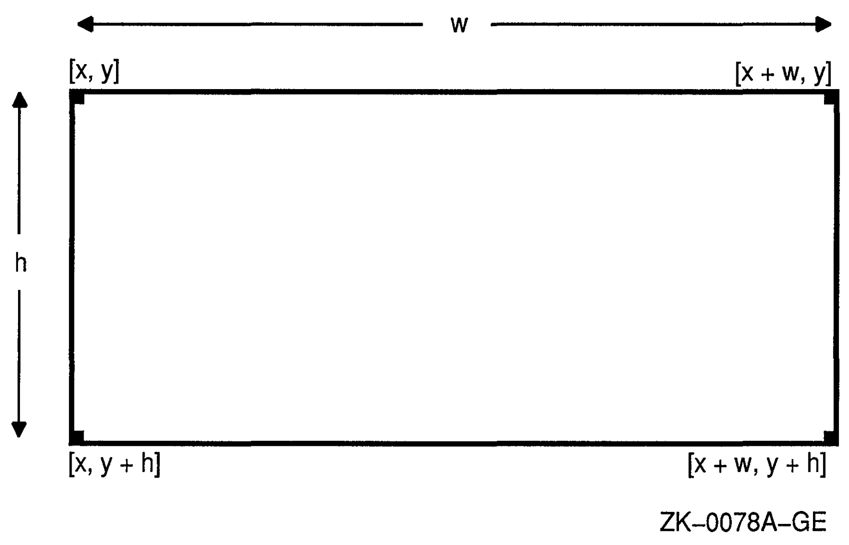
6.4.1 Drawing Rectangles

To draw a single rectangle, use the DRAW RECTANGLE routine, specifying the coordinates of the upper left corner and the dimensions of the rectangle, as in the following:

```
int x=50
int y=100;
int width=25;
int length=50;
    .
    .
    .
XDrawRectangle(display, window, gc, x, y, width, length);
```

Figure 6–3 illustrates how Xlib interprets coordinate and dimension parameters. The *x* and *y* coordinates are relative to the origin of the drawable.

Figure 6–3 Rectangle Coordinates and Dimensions

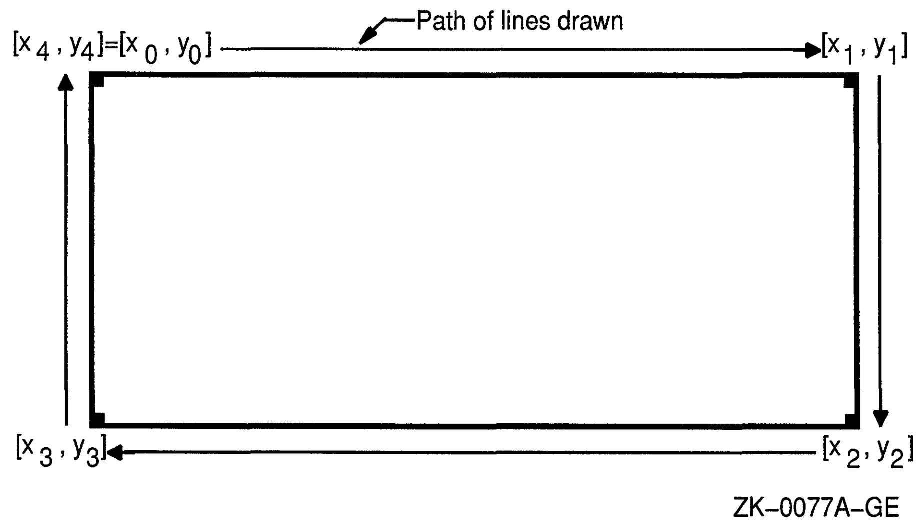


To draw multiple rectangles, use the following method:

1 Define an array of rectangles using the rectangle data structure.
2 Call the DRAW RECTANGLES routine, specifying the array that defines rectangle origin, width, and height, and the number of array elements.

The server draws each rectangle as shown in Figure 6–4.

Figure 6–4 Rectangle Drawing



For a specified rectangle, the server draws each pixel only once. If rectangles intersect, the server draws intersecting pixels multiple times.

Xlib includes the rectangle data structure to enable clients to define an array of rectangles easily. The following illustrates the data structure:

```
typedef struct {
    short x, y;
    unsigned short width, height;
} XRectangle;
```

Table 6–3 describes the members of the rectangle data structure.

Table 6–3 Rectangle Data Structure Members

Member Name	Contents
x	Defines the x value of the rectangle origin
y	Defines the y value of the rectangle origin
width	Defines the width of the rectangle
height	Defines the height of the rectangle

When drawing either single or multiple rectangles, the server refers to the following members of the GC data structure to define rectangle characteristics:

Function	Plane mask
Foreground	Background
Line width	Line style
Join style	Fill style
Tile	Stipple
Tile/stipple x origin	Tile/stipple y origin
Subwindow mode	Clip x origin
Clip y origin	Clip mask
Dash offset	Dashes

Chapter 4 describes the GC data structure members.

Example 6–3 illustrates using the DRAW RECTANGLES routine. Figure 6–5 shows the resulting output.

Example 6–3 Drawing Multiple Rectangles

```
    /* Create window win on                            *
     *  display dpy, defined as follows:               *
     *     Position: x = 100,y = 100                   *
     *     Width = 600                                 *
     *     Height = 600                                *
     * gc refers to the graphics context               */
    .
    .
    .
/****************** doHandleEvents **********************/
static void doHandleEvents( )
{
    XEvent event;

    for ( ; ; ) {
        XNextEvent(dpy, &event);
        switch (event.type) {
            case Expose:              doExpose(&event); break;
            case ButtonPress:         doButtonPress(&event); break;
        }
    }
}

/***** Write a message *****/
❶static void doExpose(eventP)
XEvent *eventP;
{
    char message1 [ ] = {"To draw multiple rectangles, click MB1"};
    char message2 [ ] = {"To exit, click MB2"};

    XDrawImageString(dpy, win, gc, 150, 25, message1, strlen(message1));
    XDrawImageString(dpy, win, gc, 150, 50, message2, strlen(message2));
}

/***** Draw the rectangles  *****/
static void doButtonPress(eventP)
XEvent *eventP;
{
#define REC_CNT 40
#define STEP 15
    XRectangle rec_arr[REC_CNT];
    int i;
❷   if (eventP->xbutton.button == Button2) sys$exit (1);

    for (i=0;i<REC_CNT;i++) {
        rec_arr[i].x = STEP * i;
        rec_arr[i].y = STEP * i;
        rec_arr[i].width = STEP*2;
        rec_arr[i].height = STEP*3;
    }

❸    XDrawRectangles(dpy, win, gc, &rec_arr, REC_CNT);
}
```

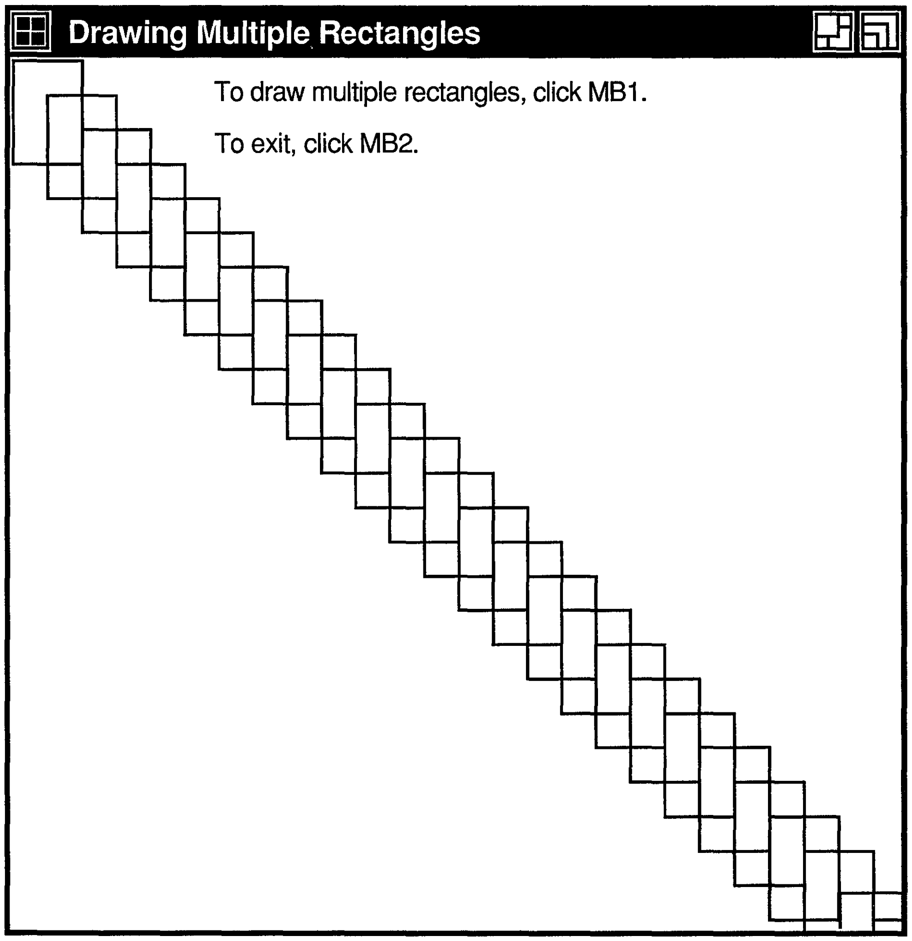
❶ When the client receives notification that the server has mapped the window, the *doExpose* routine writes two messages into the window. For information about using the DRAW IMAGE STRING routine, see Chapter 8.

❷ If the user clicks any mouse button, the client calls the *doButtonPress* routine. If the user clicks MB1, the client draws rectangles defined in the initialization loop. If the user clicks MB2, the client exits the system. The client determines which button the user has clicked by referring to the button member of the button event data structure. For more information about the button event data structure, see Chapter 9.

❸ The DRAW RECTANGLE routine has the following format:

```
XDrawRectangles(display, drawable_id, gc_id, rectangles,
            num_rectangles)
```

Figure 6–5 Rectangles Drawn Using the DRAW RECTANGLES Routine


Drawing Multiple Rectangles
To draw multiple rectangles, click MB1.
To exit, click MB2.

ZK–0105A–GE

6.4.2 Drawing Arcs

Xlib routines enable clients to draw either single or multiple arcs. To draw a single arc, use the DRAW ARC routine, specifying a rectangle that defines the boundaries of the arc and two angles that determine the start and extent of the arc, as in the following:

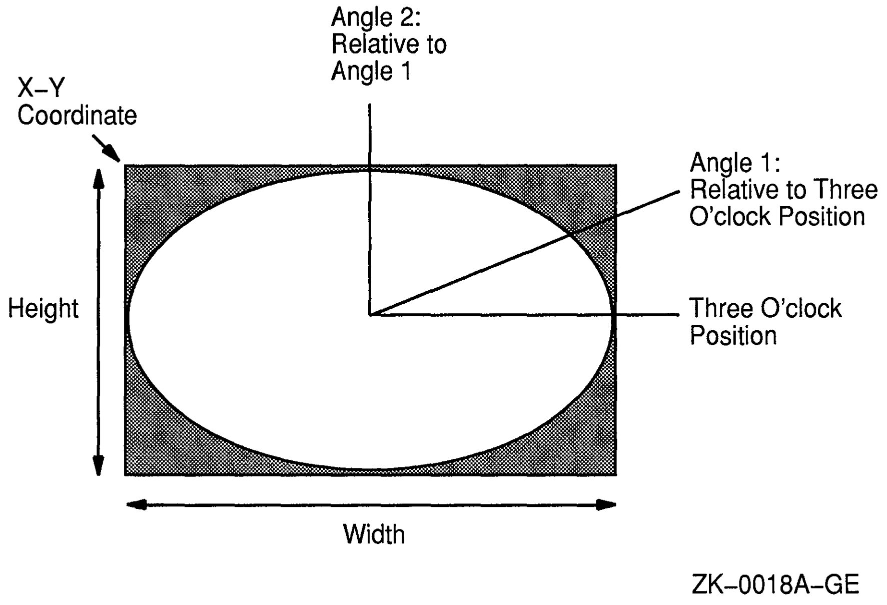
```
int x=50
int y=100;
int width=25;
int length=50;
int angle1=5760;
int angle2=5760;
    .
    .
    .
XDrawArc(display, window, gc, x, y, width, height,
     angle1, angle2);
```

The server draws an arc within a rectangle. The client specifies the upper left corner of the rectangle, relative to the origin of the drawable. The center of the rectangle is the center of the arc. The width and height of the rectangle are the major and minor axes of the arc, respectively.

Two angles specify the start and extent of the arc. The angles are signed integers in degrees scaled up by 64. For example, a client would specify a 90 degree arc as $64 * 90$ or 5760. The start of the arc is specified by the first angle, relative to the three o'clock position from the center of the rectangle. The extent of the arc is specified by the second angle, relative to the start of the arc. Positive integers indicate counterclockwise motion; negative integers indicate clockwise motion.

Figure 6–6 illustrates the relationships among the rectangle, axes, and angles that specify the arc.

Figure 6–6 Specifying an Arc


Angle 2: Relative to Angle 1
X–Y Coordinate
Angle 1: Relative to Three O'clock Position
Height
Three O'clock Position
Width
ZK–0018A–GE

For an arc specified as [$x, y, width, height, angle1, angle2$], the origin of the major and minor axes is at [$x + width/2$, $y + height/2$]. The infinitely thin path describing the entire arc intersects the horizontal axis at [$x, y + height/2$] and [$x + width$, $y + height/2$] and the vertical axis at [$x + width/2$, y] and [$x + width/2$, $y + height$]. These coordinates are not truncated to discrete coordinates if they are fractional.

The path of the arc is defined as the ideal mathematical path. For a wide line of width w, the bounding outlines for filling are given by two infinitely thin paths consisting of all points whose perpendicular distance from the path of the circle or ellipse is equal to $w/2$.

For an ellipse defined as [$x, y, width, height, angle1, angle2$], the angles must be specified in the skewed coordinate of the ellipse. The relationship between the coordinate system of the ellipse and that of a circle is specified using the following formula:

$$skewed\ angle = atan(tan(normal\ angle) * width/height) + adjust$$

The skewed angle and normal angle are expressed in radians (rather than in degrees scaled by 64) in the range [0, $2 * \pi$], where the *atan* returns a value in the range [$-\pi/2$, $\pi/2$]. The *adjust* is as follows:

- 0 for normal-angle in the range [0, $\pi/2$]
- π for a normal angle in the range [$\pi/2$, $3 * \pi/2$]
- $2 * \pi$ for a normal angle in the range [$3 * \pi/2$, $2 * \pi$]

To draw multiple arcs, use the following method:

1 Define an array of arc data structures.
2 Call the DRAW ARCS routine, specifying the array that defines the arcs and the number of array elements.

The following illustrates the arc data structure:

```
typedef struct {
    short x, y;
    unsigned short width, height;
    short angle1, angle2;
} XArc;
```

Table 6–4 describes the members of the arc data structure.

Table 6–4 Arc Data Structure Members

Member Name	Contents
x	Defines the x-coordinate value of the rectangle in which the server draws the arc
y	Defines the y-coordinate value of the rectangle in which the server draws the arc
width	Defines the major axis of the arc
height	Defines the minor axis of the arc
angle1	Defines the starting point of the arc relative to the 3-o'clock position from the center of the rectangle
angle2	Defines the extent of the arc relative to the starting point

When drawing either single or multiple arcs, the server refers to the following members of the GC data structure to define arc characteristics:

Function	Plane mask
Foreground	Background
Line width	Line style
Join style	Cap style
Fill style	Tile
Tile/stipple x origin	Tile/stipple y origin
Clip x origin	Clip y origin
Clip mask	Dash offset
Dashes	Stipple
Subwindow mode	

Chapter 4 describes the GC data structure members.

If the last point in one arc coincides with the first point in the following arc, the two arcs join. If the first point in the first arc coincides with the last point in the last arc, the two arcs join.

If two arcs join, the line width is greater than zero, and the arcs intersect, the server draws all pixels only once. Otherwise, it may draw intersecting pixels multiple times.

Example 6–4 illustrates using the DRAW ARCS routine.

Example 6–4 Drawing Multiple Arcs

```
    /* Create window win on                                  *
     *  display dpy, defined as follows:                     *
     *     Position: x = 100,y = 100                         *
     *     Width = 600                                       *
     *     Height = 600                                      *
     * gc refers to the GRAPHICS CONTEXT                     */
   .
   .
   .
/****************** doHandleEvents **********************/
static void doHandleEvents( )
{
    XEvent event;

    for ( ; ; ) {
        XNextEvent(dpy, &event);
        switch (event.type) {
            case Expose:                doExpose(&event); break;
            case ButtonPress:           doButtonPress(&event); break;
        }
    }
}

/***** Write a message *****/
static void doExpose(eventP)
XEvent *eventP;
{
    char message1[ ] = {"To create arcs, click MB1"};
    char message2[ ] = {"Each click creates a new circle of arcs."};
    char message3[ ] = {"To exit, click MB2"};

    XDrawImageString(dpy, win, gc, 150, 25, message1, strlen(message1));
    XDrawImageString(dpy, win, gc, 150, 50, message2, strlen(message2));
    XDrawImageString(dpy, win, gc, 150, 75, message3, strlen(message3));
}

/***** Draw the arcs *****/
static void doButtonPress(eventP)
XEvent *eventP;
{
#define ARC_CNT 16
#define RADIUS 50
#define INNER_RADIUS  20
    XArc arc_arr[ARC_CNT];
    int i;
➊   int x = eventP->xbutton.x;
    int y = eventP->xbutton.y;

    if (eventP->xbutton.button == Button2) sys$exit (1);
```

(continued on next page)

Example 6–4 (Cont.) Drawing Multiple Arcs

```
    for (i=0;i<ARC_CNT;i++) {
        arc_arr[i].angle1 = (64*360)/ARC_CNT * i;
        arc_arr[i].angle2 = (64*360)/ARC_CNT*3;
        arc_arr[i].width = RADIUS*2;
        arc_arr[i].height = RADIUS*2;
        arc_arr[i].x = x - RADIUS + sin(2*3.14159/ARC_CNT*i) * INNER_RADIUS;
        arc_arr[i].y = y - RADIUS + cos(2*3.14159/ARC_CNT*i) * INNER_RADIUS;
    }
❷  XDrawArcs(dpy, win, gc, &arc_arr, ARC_CNT);
}
```

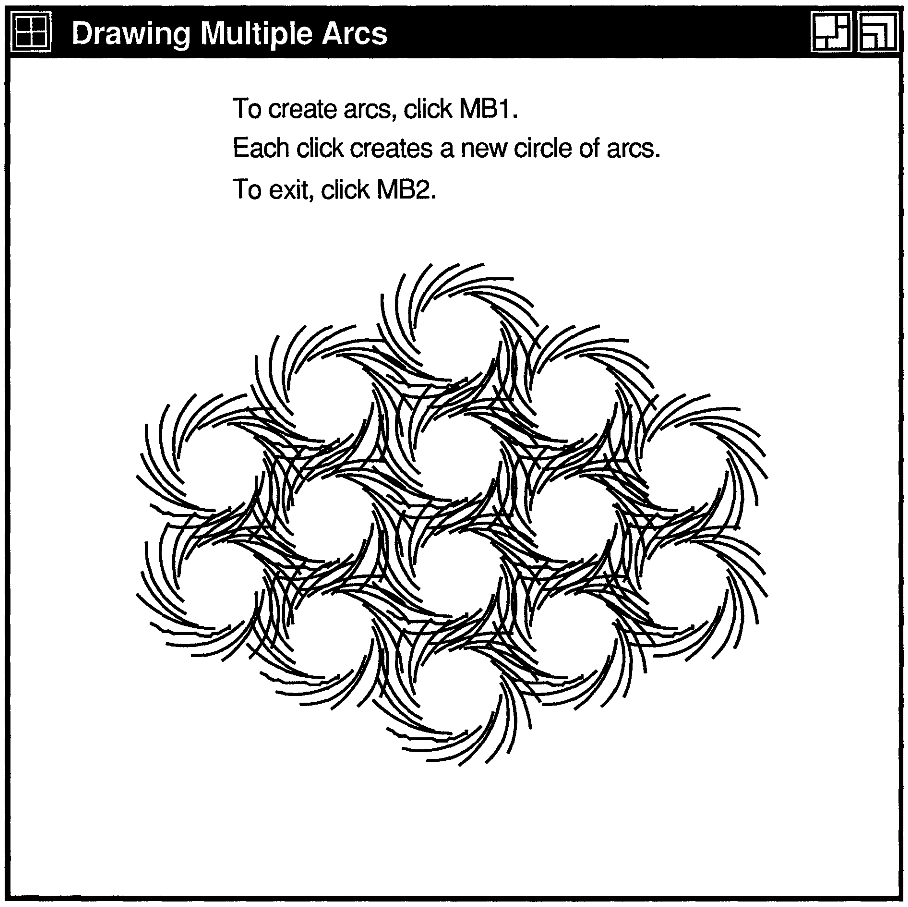
❶ The *x* and *y* variables specify the upper left corner of the rectangle that defines the boundary of the arc. The client determines the rectangle coordinates by taking the values of the **x** and **y** arguments from the button event data structure. Because these values indicate the position of the cursor when the user clicks the mouse button, the server draws the arcs relative to the position of the cursor. For more information about the button event data structure, see Chapter 9.

❷ The DRAW ARCS routine has the following format:

```
XDrawArcs(display,drawable_id,gc_id,arcs,num_arcs)
```

Figure 6–7 illustrates the resulting output.

Figure 6–7 Multiple Arcs Drawn Using the DRAW ARCS Routine



To create arcs, click MB1.
Each click creates a new circle of arcs.
To exit, click MB2.

ZK–0106A–GE

6.5 Filling Areas

This section describes using Xlib routines to fill single rectangles, arcs, and polygons, and multiple rectangles and arcs.

6.5.1 Filling Rectangles and Arcs

The FILL RECTANGLE, FILL RECTANGLES, FILL ARC, and FILL ARCS routines create single and multiple rectangles or arcs and fill them using the fill style the client specifies in a graphics context data structure.

The method of calling the fill routines is identical to that for drawing rectangles and arcs. For example, to create rectangles filled solidly with foreground color in Example 6–3, the client needs only to call the FILL RECTANGLES routine instead of DRAW RECTANGLES. The default value of the GC data structure fill style member is solid. If the client were to specify a tile or stipple for filling the rectangles, the client would have to change the graphics context used by the FILL RECTANGLES routine.

The server refers to the following members of the GC data structure to define characteristics of the rectangles and arcs it fills:

Function	Plane mask
Foreground	Background
Fill style	Tile
Stipple	Subwindow mode
Tile/stipple x origin	Tile/stipple y origin
Clip x origin	Clip y origin
Clip mask	

Additionally, the server refers to the arc mode member if filling arcs.

For information about using graphics context, see Chapter 4.

6.5.2 Filling a Polygon

To fill a polygon, use the following method:

1 Define an array of point data structures.
2 Call the FILL POLYGON routine, specifying the array that defines the points of the polygon, the number of points the server is to draw, the shape of the polygon, and the coordinate system the server is to use. The server draws the points in the order specified by the array.

See Section 6.3.1 for an illustration of the point data structure.

To improve performance, clients can specify whether the shape of the polygon is complex, convex, or nonconvex, as follows:

- Specify the constant **Complex** as the **shape** argument if the path that draws the polygon may intersect itself.
- Specify the constant **Convex** if the path that draws the shape is wholly convex. If a client specifies **Convex** for a path that is not convex, the results are undefined.
- Specify the constant **Nonconvex** as the **shape** argument if the path does not intersect itself, but the shape is not wholly convex. If a client specifies **Nonconvex** for a path that intersects itself, the results are undefined.

When filling the polygon, the server draws each pixel only once.

The server determines the location of points as follows:

- If the client specifies the constant **CoordModeOrigin**, the server defines all points in the array relative to the origin of the drawable.
- If the client specifies the constant **CoordModePrevious**, the server defines the coordinates of the first point in the array relative to the origin of the drawable, and the coordinates of each subsequent point relative to the point preceding it in the array.

If the last point does not coincide with the first point, the server closes the polygon automatically.

The server refers to the following members of the GC data structure to define the characteristics of the polygon it fills:

Function	Plane mask
Foreground	Fill style
Fill rule (if polygon is complex)	Tile
Tile/stipple x origin	Tile/stipple y origin
Clip x origin	Clip y origin
Subwindow mode	Clip mask
Stipple	Background

Chapter 4 describes GC data structure members.

Example 6–5 uses the FILL POLYGON routine to draw and fill the star created in Example 6–2.

Example 6–5 Filling a Polygon

```
    /* Create window win on                               *
     *  display dpy, defined as follows:                  *
     *     Position: x = 100,y = 100                      *
     *     Width = 600                                    *
     *     Height = 600                                   *
     * gc refers the graphics context                     */
   .
   .
   .
/****************** doHandleEvents ***********************/
static void doHandleEvents( )
{
    XEvent event;

    for ( ; ; ) {
        XNextEvent(dpy, &event);
        switch (event.type) {
            case Expose:                doExpose(&event); break;
        }
    }
}
/***** doExpose ****/
static void doExpose(eventP)
XEvent *eventP;
{
    XPoint pt_arr[6];
```

(continued on next page)

Example 6–5 (Cont.) Filling a Polygon

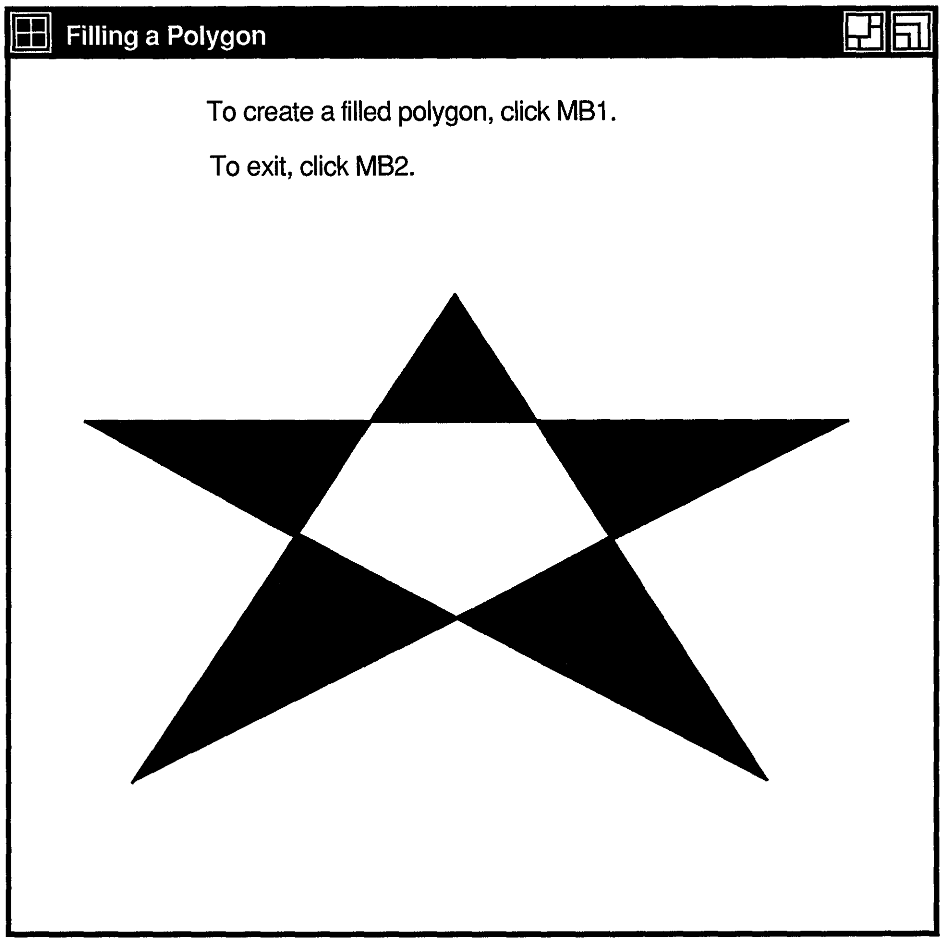
```
❶  pt_arr[0].x = 75;
   pt_arr[0].y = 500;
   pt_arr[1].x = 300;
   pt_arr[1].y = 100;
   pt_arr[2].x = 525;
   pt_arr[2].y = 500;
   pt_arr[3].x = 50;
   pt_arr[3].y = 225;
   pt_arr[4].x = 575;
   pt_arr[4].y = 225;
   pt_arr[5].x = 75;
   pt_arr[5].y = 500;

❷   XFillPolygon(dpy, win, gc, &pt_arr, 6, Complex, CoordModeOrigin);
}
    .
    .
    .
```

❶ Use an array of point data structures to specify the points that define the polygon.

❷ The call to fill the polygon refers to a graphics context (*gc*), which the client has previously defined, and an array of point data structures. The constant **Complex** indicates that the path of the line that draws the polygon intersects itself. The constant **CoordModeOrigin** indicates that all points are relative to the origin of *win* (100,100).

Figure 6–8 illustrates the resulting output.

Figure 6–8 Filled Star Created Using the FILL POLYGON Routine



ZK–0158A–GE

6.6 Clearing and Copying Areas

Xlib includes routines that enable clients to clear or copy a specified area of a drawable. Because pixmaps do not have defined backgrounds, clients clearing an area of a pixmap must use the FILL RECTANGLE routine described in Section 6.5.1. For more information about pixmaps, see Chapter 7.

This section describes how to clear windows and copy areas of windows and pixmaps.

6.6.1 Clearing Window Areas

To clear an area of a window, use the CLEAR AREA or CLEAR WINDOW routine. The CLEAR AREA routine clears a specified area and generates an exposure event, if the client directs the server to do so.

The CLEAR WINDOW routine clears the entire area of the specified window. If the window has a defined background tile, the window is retiled. If the window has no defined background, the server does not change the window contents.

Example 6–6 illustrates clearing a window.

Example 6–6 Clearing a Window

```
    .
    .
    .
/***** Draw multiple arcs  *****/
static void doButtonPress(eventP)
XEvent *eventP;
{
#define ARC_CNT 16
#define RADIUS 50
#define INNER_RADIUS  20
    XArc arc_arr[ARC_CNT];
    int i;
    int x = eventP->xbutton.x;
    int y = eventP->xbutton.y;

    if (eventP->xbutton.button == Button2) sys$exit (1);
    if (eventP->xbutton.button == Button3)
    {
        XClearWindow(dpy, win);
        return;
    }

    for (i=0;i<ARC_CNT;i++) {
        arc_arr[i].angle1 = (64*360)/ARC_CNT * i;
        arc_arr[i].angle2 = (64*360)/ARC_CNT*3;
        arc_arr[i].width = RADIUS*2;
        arc_arr[i].height = RADIUS*2;
        arc_arr[i].x = x - RADIUS + sin(2*3.14159/ARC_CNT*i) * INNER_RADIUS;
        arc_arr[i].y = y - RADIUS + cos(2*3.14159/ARC_CNT*i) * INNER_RADIUS;
    }

    XDrawArcs(dpy, win, gc, &arc_arr, ARC_CNT);
}
```

The example modifies the *doButtonPress* routine of Example 6–4 to clear the window when the user clicks MB3.

If clearing multiple areas, using the FILL RECTANGLES routine is faster than using the CLEAR WINDOW or CLEAR AREA routine. To clear multiple areas on a monochrome screen, first set the function member of the GC data structure to the value specified by the constant **GXclear**. Then call the FILL RECTANGLES routine. If the screen is a color type, set the value of the background to the background of the window before calling FILL RECTANGLES.

6.6.2 Copying Areas of Windows and Pixmaps

Xlib includes the COPY AREA and COPY PLANE routines to enable clients to copy a rectangular area defined on one window or pixmap (the source) to an area of another window or pixmap (the destination). COPY AREA copies areas between drawables of the same root and depth. COPY PLANE copies a single bit plane of the specified drawable to another drawable, regardless of their depths. The bit plane is treated as a stipple with a fill style of **FillOpaqueStippled**. Both drawables must have the same root window.

The server refers to the following members of the GC data structure when copying areas and planes:

Function	Plane mask
Clip x origin	Clip y origin
Subwindow mode	Clip mask
Graphics exposures	

If the client calls COPY AREA or COPY PLANES, the server also refers to the graphics exposures member of the GC data structure. If the client calls the COPY PLANES routine, the server additionally refers to the foreground and background members.

6.7 Defining Regions

A **region** is an arbitrarily defined area within which graphics drawing is clipped. In other words, clipping regions are portions of either windows or pixmaps in which clients can restrict output. As Chapter 4 notes, the SET CLIP MASK, SET CLIP ORIGIN, and SET CLIP RECTANGLES routines define clipping regions. Xlib provides other, more convenient, routines that enable clients to define regions and associate them with drawables without having to change graphics context values directly.

This section describes how to create and manage clipping using Xlib region routines.

6.7.1 Creating Regions

Xlib includes the CREATE REGION and POLYGON REGION routines for creating regions. CREATE REGION creates an empty region. POLYGON REGION creates a region defined by an array of points.

Example 6–7 illustrates using POLYGON REGION to create a star-shaped region. Using the DRAW ARCS routine of Example 6–4, the program limits arc drawing to the star region.

Example 6–7 Defining a Region Using the POLYGON REGION Routine

```
    /* Create window win on                                    *
     *  display dpy, defined as follows:                       *
     *     Position: x = 100,y = 100                           *
     *     Width = 600                                         *
     *     Height = 600                                        *
     * gc refers to the graphics context                       */
    .
    .
    .
/******** Create the graphics context *********/
static void doCreateGraphicsContext( )
{
    XPoint pt_arr[NUM_PTS];
    XGCValues xgcv;

❶   pt_arr[0].x = 75;
    pt_arr[0].y = 500;
    pt_arr[1].x = 300;
    pt_arr[1].y = 100;
    pt_arr[2].x = 525;
    pt_arr[2].y = 500;
    pt_arr[3].x = 50;
    pt_arr[3].y = 225;
    pt_arr[4].x = 575;
    pt_arr[4].y = 225;
    pt_arr[5].x = 75;
    pt_arr[5].y = 500;

    /* Create graphics context. */

    xgcv.foreground = doDefineColor(2);
    xgcv.background = doDefineColor(3);

    gc = XCreateGC(dpy, win, GCForeground | GCBackground, &xgcv);
❷   star_region = XPolygonRegion(&pt_arr, NUM_PTS, WindingRule);
}
    .
    .
    .
/****************** doHandleEvents **********************/
static void doHandleEvents( )
{
    XEvent event;

    for ( ; ; ) {
        XNextEvent(dpy, &event);
        switch (event.type) {
            case Expose:             doExpose(&event); break;
            case ButtonPress:        doButtonPress(&event); break;
        }
    }
}
```

(continued on next page)

Example 6–7 (Cont.) Defining a Region Using the POLYGON REGION Routine

```
/***** Write a message *****/
static void doExpose(eventP)
XEvent *eventP;
{
    char message1[ ] = {"To create arcs in a region, click MB1"};
    char message2[ ] = {"Each click creates a new circle of arcs."};
    char message3[ ] = {"To exit, click MB2"};

    XDrawImageString(dpy, win, gc, 150, 25, message1, strlen(message1));
    XDrawImageString(dpy, win, gc, 150, 50, message2, strlen(message2));
    XDrawImageString(dpy, win, gc, 150, 75, message3, strlen(message3));
}

/***** Draw the arcs *****/
static void doButtonPress(eventP)
XEvent *eventP;
{
#define ARC_CNT 16
#define RADIUS 50
#define INNER_RADIUS  20
    XArc arc_arr[ARC_CNT];
    int i;
    int x = eventP->xbutton.x;
    int y = eventP->xbutton.y;

    if (eventP->xbutton.button == Button2) sys$exit (1);

❸   XSetRegion(dpy, gc, star_region);
    for (i=0;i<ARC_CNT;i++) {
        arc_arr[i].angle1 = (64*360)/ARC_CNT * i;
        arc_arr[i].angle2 = (64*360)/ARC_CNT*3;
        arc_arr[i].width = RADIUS*2;
        arc_arr[i].height = RADIUS*2;
        arc_arr[i].x = x - RADIUS + sin(2*3.14159/ARC_CNT*i) * INNER_RADIUS;
        arc_arr[i].y = y - RADIUS + cos(2*3.14159/ARC_CNT*i) * INNER_RADIUS;
    }

    XDrawArcs(dpy, win, gc, &arc_arr, ARC_CNT);
}
```

❶ Define an array of point data structures to define the clipping region.

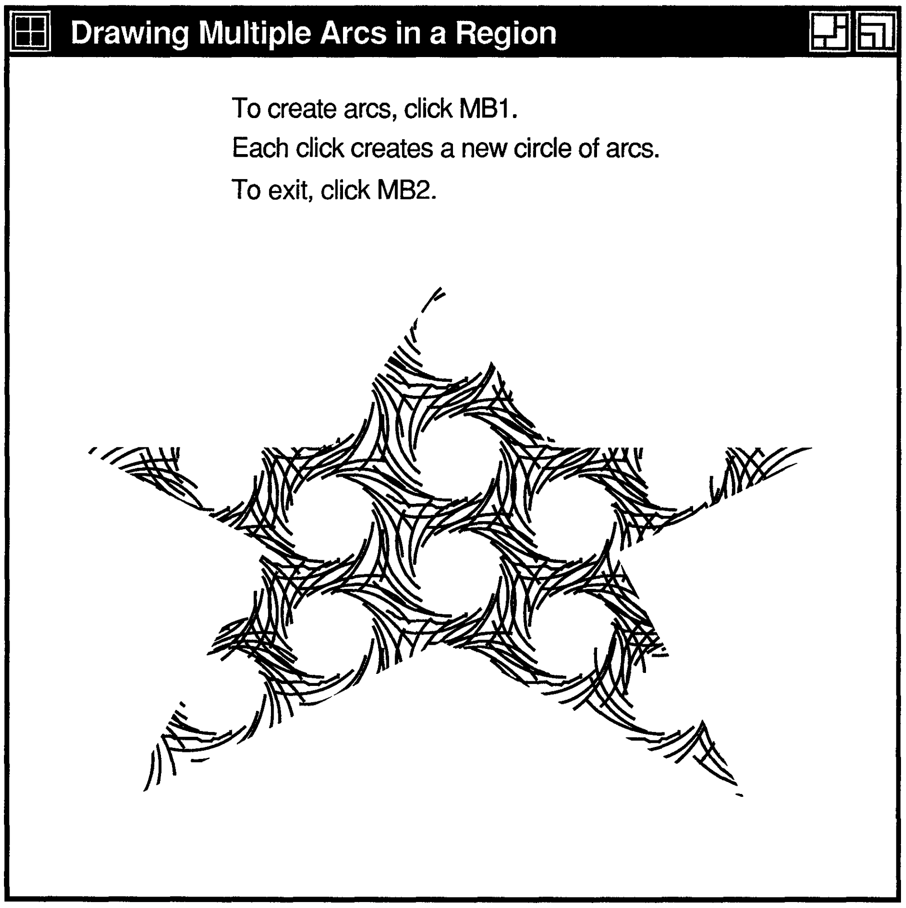
❷ Define the clipping region. Note that defining the region does not associate it with a graphics context.

Fill rule can be either even odd rule or winding rule. For more information about fill rule, see Chapter 4.

❸ Associate the region with a graphics context. The association sets fields in the specified GC data structure that control clipping. Drawables that refer to the GC data structure have output clipped to the region.

Figure 6–9 illustrates sample output from the program.

Figure 6–9 Arcs Drawn Within a Region



Drawing Multiple Arcs in a Region

To create arcs, click MB1.
Each click creates a new circle of arcs.
To exit, click MB2.

ZK–0323A–GE

6.7.2 Managing Regions

Xlib includes routines that enable clients to do the following:

- Move and shrink a region
- Compute the intersection, union, and results of two regions
- Determine if regions are empty or equal
- Locate a point or rectangle within a region

Table 6–5 lists and describes Xlib routines that manage regions.

Table 6–5 Routines for Managing Regions

Routine	Description
Moving and Shrinking	
OFFSET REGION	Moves a region a specified amount
SHRINK REGION	Reduces a region a specified amount
Computing	
INTERSECT REGION	Computes the intersection of two regions
UNION REGION	Computes the union of two regions
SUBTRACT REGION	Subtracts two regions
XOR REGION	Calculates the difference between the union and intersection of two regions
Determining if Regions are Empty or Equal	
EMPTY REGION	Determines if a region is empty
EQUAL REGION	Determines if two regions have the same offset, size, and shape
Locating a Point or Rectangle Within a Region	
POINT IN REGION	Determines if a point is within a region
RECT IN REGION	Determines if a rectangle is within a region

Example 6–8 illustrates creating a region from the intersection of two others.

Example 6–8 Defining the Intersection of Two Regions

```
Pixmap pixmap1, pixmap2, pixmap3;
Region region1, region2, region3;
    .
    .
    .
/***************** doInitialize *************************/
static void doInitialize( )
{
    dpy = XOpenDisplay(0);

    screen = XDefaultScreenOfDisplay(dpy);

    doCreateWindows( );

    doCreateGraphicsContext( );

    doCreatePixmap( );

    doCreateRegion( );

    doWMHints( );

    doMapWindows( );
}
    .
    .
    .
/******* doCreatePixmap *********/
❶static void doCreatePixmap( )
{
    pixmap1 = XCreatePixmap(dpy, win, pixWidth, pixHeight,
              DefaultDepthOfScreen(screen));
    pixmap2 = XCreatePixmap(dpy, win, pixWidth, pixHeight,
              DefaultDepthOfScreen(screen));
    pixmap3 = XCreatePixmap(dpy, win, pixWidth, pixHeight,
              DefaultDepthOfScreen(screen));

    /* Set the pixmap background */
    XFillRectangle(dpy, pixmap1, gc, 0, 0, pixWidth, pixHeight);
    XFillRectangle(dpy, pixmap2, gc, 0, 0, pixWidth, pixHeight);
    XFillRectangle(dpy, pixmap3, gc, 0, 0, pixWidth, pixHeight);

    /* Redefine foreground value for line drawing and text */
    XSetForeground(dpy, gc, doDefineColor(2));

    /* Draw Line into the pixmap */
    XDrawLine(dpy, pixmap1, gc, 0, 4, 0, 8);
    XDrawLine(dpy, pixmap2, gc, 4, 0, 8, 0);
    XDrawLine(dpy, pixmap3, gc, 0, 4, 0, 8);
    XDrawLine(dpy, pixmap3, gc, 4, 0, 8, 0);

}
/******* doCreateRegion *********/
static void doCreateRegion( )
{
❷   XPoint pt_arr_1[num_pts], pt_arr_2[num_pts];
```

(continued on next page)

Example 6–8 (Cont.) Defining the Intersection of Two Regions

```
    pt_arr_1[0].x = 200;
    pt_arr_1[0].y = 100;
    pt_arr_1[1].x = 50;
    pt_arr_1[1].y = 300;
    pt_arr_1[2].x = 200;
    pt_arr_1[2].y = 500;
    pt_arr_1[3].x = 350;
    pt_arr_1[3].y = 300;

    pt_arr_2[0].x = 400;
    pt_arr_2[0].y = 100;
    pt_arr_2[1].x = 250;
    pt_arr_2[1].y = 300;
    pt_arr_2[2].x = 400;
    pt_arr_2[2].y = 500;
    pt_arr_2[3].x = 550;
    pt_arr_2[3].y = 300;

    region1 = XPolygonRegion(pt_arr_1, num_pts, WindingRule);
    region2 = XPolygonRegion(pt_arr_2, num_pts, WindingRule);

}
    .
    .
    .
/****************** doHandleEvents ***********************/
static void doHandleEvents( )
{
    XEvent event;

    for ( ; ; ) {
        XNextEvent(dpy, &event);
        switch (event.type) {
            case Expose:                doExpose(&event); break;
            case ButtonPress:           i++; doButtonPress(&event); break;
        }
    }
}
/***** Write a message *****/
static void doExpose(eventP)
XEvent *eventP;
{
    char message1[ ] = {"To map regions click MB1 three times."};
    char message2[ ] = {"To exit, click MB2."};

    XDrawImageString(dpy, win, gc, 150, 25, message1, strlen(message1));
    XDrawImageString(dpy, win, gc, 150, 50, message2, strlen(message2));
}
/***** Map the regions when the button is pressed *****/
static void doButtonPress(eventP)
XEvent *eventP;
{
    char message3[ ] = {"That's it! Click MB2 to exit."};

    if (eventP->xbutton.button == Button2) sys$exit (1);
    if (i == 1){

       /* Redefine the fill style for stippling */
❸          XSetFillStyle(dpy, gc, FillTiled);
```

(continued on next page)

Example 6–8 (Cont.) Defining the Intersection of Two Regions

```
         XClearWindow(dpy, win);
         XSetTile(dpy, gc, pixmap1);
❹        XSetRegion(dpy, gc, region1);
❺        XFillRectangle(dpy, win, gc, xOrigin, yOrigin, winW, winH);
         }
     else if (i == 2){
❻        XClearWindow(dpy, win);
         XSetTile(dpy, gc, pixmap2);
         XSetRegion(dpy, gc, region2);
         XFillRectangle(dpy, win, gc, xOrigin, yOrigin, winW, winH);
         }
     else if (i == 3){
         XClearWindow(dpy, win);
❼        region3 = XCreateRegion();
         XIntersectRegion(region1, region2, region3);
         XSetTile(dpy, gc, pixmap3);
         XSetRegion(dpy, gc, region3);
         XFillRectangle(dpy, win, gc, xOrigin, yOrigin, winW, winH);
         }
     else{
         /* To draw text, redefine the fill style as solid */
❽        XSetFillStyle(dpy, gc, FillSolid);
         XDrawImageString(dpy, win, gc, 150, 50, message3, strlen(message3));
         }
}
```

❶ Pixmaps are used to tile the window with horizontal, vertical, and cross-hatched lines. For information about pixmaps, see Chapter 7.

❷ Arrays of point data structures define two regions.

❸ After writing messages in the window, the fill style defined in the GC data structure is changed to tile the window with pixmaps. The subsequent call to SET TILE defines one of the three pixmaps created earlier as the window background pixmap. For information about fill styles and tiling, see Chapter 4.

❹ The SET REGION routine specifies the clipping region in the graphics context. The region defined by *pt_arr1* is first specified.

❺ FILL RECTANGLE repaints the window, filling it with the tiling pattern defined in *pixmap1*. Tiling is restricted to the region defined by *region1*.

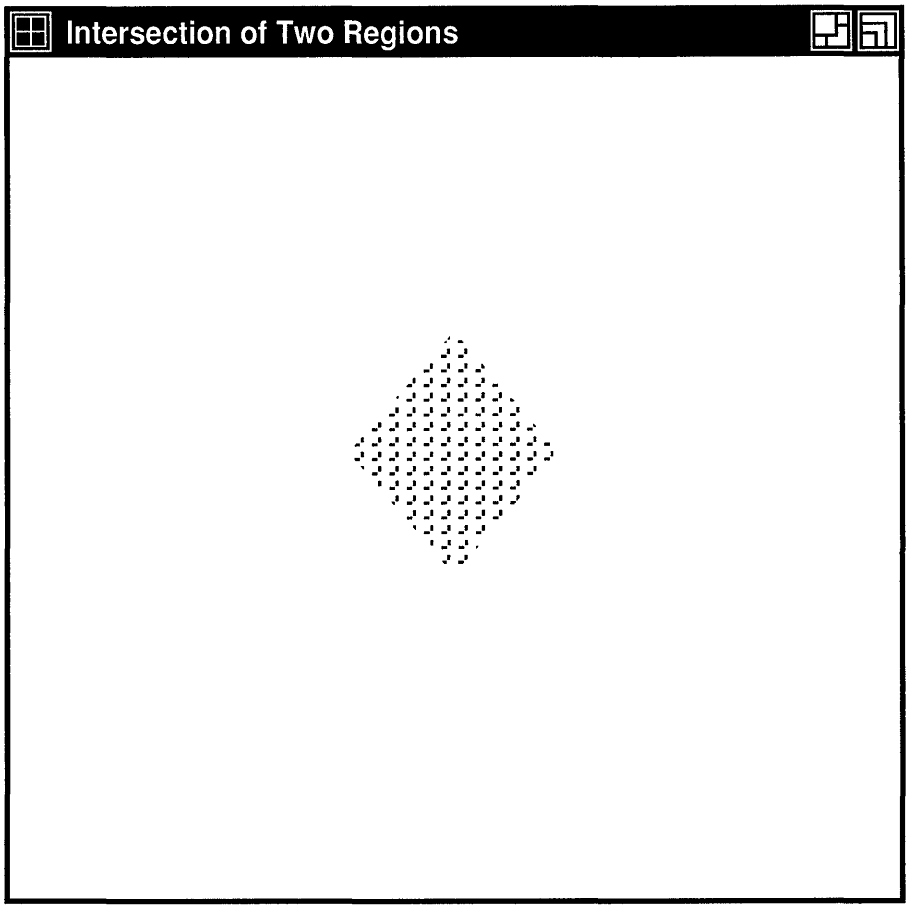
❻ Before specifying a new tiling pattern and region, the window is cleared.

❼ CREATE REGION creates an empty region and returns an identifier, *region3*. Xlib returns the results of intersecting *region1* and *region2* to *region3*.

❽ Before displaying a final message in the window, the fill style is redefined to solid to enable text writing.

Figure 6–10 illustrates the output from the program.

Figure 6–10 Intersection of Two Regions



ZK–0322A–GE

6.8 Defining Cursors

A **cursor** is a bit image on the screen that indicates either the movement of a pointing device or the place where text will next appear. Xlib enables clients to associate a cursor with each window they create. After making the association between cursor and window, the cursor is visible whenever it is in the window. If the cursor indicates movement of a pointing device, the movement of the cursor in the window automatically reflects the movement of the device.

Xlib and VMS DECwindows provide fonts of predefined cursors. Clients that want to create their own cursors can either define a font of shapes and masks or create cursors using pixmaps.

This section describes the following:

- Creating cursors using the Xlib cursor font, a font of shapes and masks, and pixmaps
- Associating cursors with windows
- Managing cursors
- Freeing memory allocated to cursors when clients no longer need them

6.8.1 Creating Cursors

Xlib enables clients to use predefined cursors or to create their own cursors. To create a predefined Xlib cursor, use the CREATE FONT CURSOR routine. Xlib cursors are predefined in DECW$INCLUDE:CURSORFONT.H. Table 6–6 lists the constants that refer to the predefined Xlib cursors.

Table 6–6 Predefined Xlib Cursors

XC_X_cursor	XC_arrow
XC_based_arrow_down	XC_based_arrow_up
XC_boat	XC_bogosity
XC_bottom_left_corner	XC_bottom_right_corner
XC_bottom_side	XC_bottom_tee
XC_box_spiral	XC_center_ptr
XC_circle	XC_clock
XC_coffee_mug	XC_cross
XC_cross_reverse	XC_crosshair
XC_diamond_cross	XC_dot
XC_dotbox	XC_double_arrow
XC_draft_large	XC_draft_small
XC_draped_box	XC_exchange
XC_fleur	XC_gobbler
XC_gumby	XC_hand1
XC_hand2	XC_heart
XC_icon	XC_iron_cross
XC_left_ptr	XC_left_side
XC_left_tee	XC_leftbutton
XC_ll_angle	XC_lr_angle
XC_man	XC_middlebutton
XC_mouse	XC_pencil
XC_pirate	XC_plus

(continued on next page)

Table 6–6 (Cont.) Predefined Xlib Cursors

XC_question_arrow	XC_right_ptr
XC_right_side	XC_right_tee
XC_rightbutton	XC_rtl_logo
XC_sailboat	XC_sb_down_arrow
XC_sb_h_double_arrow	XC_sb_left_arrow
XC_sb_right_arrow	XC_sb_up_arrow
XC_sb_v_double_arrow	XC_shuttle
XC_sizing	XC_spider
XC_spraycan	XC_star
XC_target	XC_tcross
XC_top_left_arrow	XC_top_left_corner
XC_top_right_corner	XC_top_side
XC_top_tee	XC_trek
XC_ul_angle	XC_umbrella
XC_ur_angle	XC_watch
XC_xterm	

The following example creates a sailboat cursor, one of the predefined Xlib cursors, and associates the cursor with a window:

```
Cursor fontcursor;
    .
    .
    .
fontcursor = XCreateFontCursor(dpy, XC_sailboat);
XDefineCursor(dpy, win, fontcursor);
```

The DEFINE CURSOR routine makes the sailboat cursor automatically visible when the pointer is in window *win*.

To create a predefined VMS DECwindows cursor, use the CREATE GLYPH CURSOR routine. VMS DECwindows cursors are predefined in SYS$LIBRARY:DECW$CURSOR.H. Table 6–7 lists the constants that refer to the predefined VMS DECwindows cursors.

Table 6–7 Predefined VMS DECwindows Cursors

decw$c_select_cursor	decw$c_leftselect_cursor
decw$c_help_select_cursor	decw$c_wait_cursor
decw$c_inactive_cursor	decw$c_resize_cursor
decw$c_vpane_cursor	decw$c_hpane_cursor
decw$c_text_insertion_cursor	decw$c_text_insertion_bl_cursor
decw$c_cross_hair_cursor	decw$c_draw_cursor

(continued on next page)

Table 6–7 (Cont.) Predefined VMS DECwindows Cursors

decw$c_pencil_cursor	decw$c_rpencil_cursor
decw$c_center_cursor	decw$c_rightselect_cursor
decw$c_wselect_cursor	decw$c_eselect_cursor
decw$c_x_cursor	decw$c_circle_cursor
decw$c_mouse_cursor	decw$c_lpencil_cursor
decw$c_leftgrab_cursor	decw$c_grabhand_cursor
decw$c_rightgrab_cursor	decw$c_leftpointing_cursor
decw$c_uppointing_cursor	decw$c_rightpointing_cursor

CREATE GLYPH CURSOR selects a cursor shape and cursor mask from the VMS DECwindows cursor font, defines how the cursor appears on the screen, and assigns a unique cursor identifier. The following example illustrates creating the select cursor and associating the cursor with a window:

```
Font cursorfont
Cursor glyphcursor;
XColor forecolor, backcolor;
    .
    .
    .
cursorfont = XLoadFont(dpy, "decw$cursor");
XSetFont(dpy, gc, "decw$cursor");

glyphcursor = XCreateGlyphCursor(dpy, cursorfont, cursorfont,
     decw$c_select_cursor, decw$c_select_cursor + 1,
     &forecolor, &backcolor);
XDefineCursor(dpy, win, glyphcursor);
```

To create client-defined cursors, either create a font of cursor shapes or define cursors using pixmaps. In each case the cursor consists of the following components:

- Shape—Defines the cursor as it appears without modification in a window
- Mask—Acts as a clip mask to define how the cursor actually appears in a window
- Background color—Specifies RGB values used for the cursor background
- Foreground color—Specifies RGB values used for the cursor foreground
- Hot spot—Defines the position on the cursor that reflects movements of the pointing device

Figure 6–11 illustrates the relationship between the cursor shape and the cursor mask. The cursor shape defines the cursor as it would appear on the screen without modification. The cursor mask bits that are set to 1 select which bits of the cursor shape are actually displayed. If the mask bit has a value of 1, the corresponding shape bit is displayed whether it has a value of 1 or 0. If the mask bit has a value of 0, the corresponding shape bit is not displayed.

In the resulting cursor shape, bits with a 0 value are displayed in the specified background color; bits with a 1 value are displayed in the specified foreground color.

To create a client-defined cursor from a font of glyphs, use the CREATE GLYPH CURSOR routine, specifying the cursor and mask fonts that contain the glyphs. To create a cursor from pixmaps, use the CREATE PIXMAP CURSOR routine. The pixmaps must have a depth of one. If the depth is not one, the server generates an error.

Figure 6–11 Cursor Shape and Cursor Mask

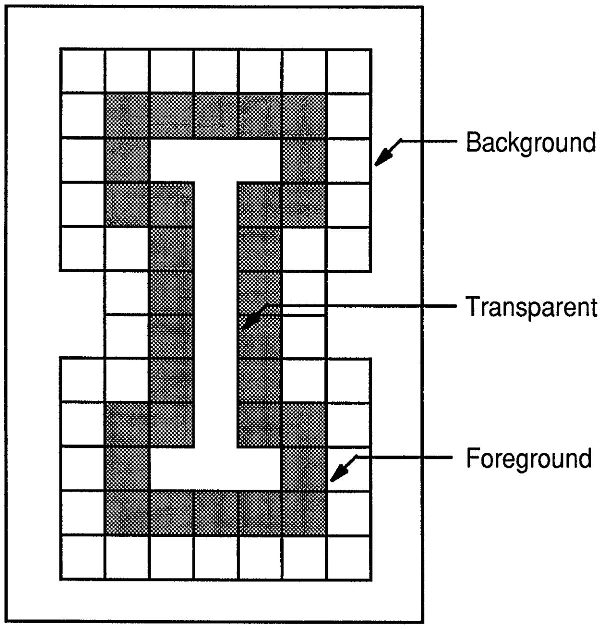
Cursor Shape

0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0
0	0	0	1	1	1	1	1	0	0	0
0	0	0	1	0	0	0	1	0	0	0
0	0	0	1	1	0	1	1	0	0	0
0	0	0	0	1	0	1	0	0	0	0
0	0	0	0	1	0	1	0	0	0	0
0	0	0	0	1	0	1	0	0	0	0
0	0	0	0	1	0	1	0	0	0	0
0	0	0	1	1	0	1	1	0	0	0
0	0	0	1	0	0	0	1	0	0	0
0	0	0	1	1	1	1	1	0	0	0
0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0

Cursor Mask

0	0	0	0	0	0	0	0	0	0	0
0	0	1	1	1	1	1	1	1	0	0
0	0	1	1	1	1	1	1	1	0	0
0	0	1	1	0	0	0	1	1	0	0
0	0	1	1	1	0	1	1	1	0	0
0	0	1	1	1	0	1	1	1	0	0
0	0	0	1	1	0	1	1	0	0	0
0	0	0	1	1	0	1	1	0	0	0
0	0	1	1	1	0	1	1	1	0	0
0	0	1	1	1	0	1	1	1	0	0
0	0	1	1	0	0	0	1	1	0	0
0	0	1	1	1	1	1	1	1	0	0
0	0	1	1	1	1	1	1	1	0	0
0	0	0	0	0	0	0	0	0	0	0

Resulting Cursor



ZK–0154A–GE

The size of the pixmap cursor must be supported by the display on which the cursor is visible. To determine the supported size closest to the size the client specifies, use the QUERY BEST CURSOR routine. Example 6–9 illustrates creating a pencil pointer cursor from two pixmaps.

Example 6–9 Creating a Pixmap Cursor

```
#include <decw$include/Xlib.h>
#include <decw$include/Xutil.h>

#define winW 600
#define winH 600
#define pencil_width 16
#define pencil_height 16
#define pencil_xhot 1
#define pencil_yhot 15

Display *dpy;
Window win;
Pixmap pixmap, pencil;
Pixmap pencil_mask;
Cursor pencil_cursor;
   .
   .
   .
static char pencil_bits[] = {
   0x0000, 0x0070, 0x0000, 0x0088, 0x0000, 0x008C, 0x0000, 0x0096,
   0x0000, 0x0069, 0x0080, 0x0030, 0x0040, 0x0010, 0x0020, 0x0008,
   0x0010, 0x0004, 0x0008, 0x0002, 0x0008, 0x0001, 0x0094, 0x0000,
   0x0064, 0x0000, 0x001E, 0x0000, 0x0006, 0x0000, 0x0000, 0x0000};

static char pencil_mask_bits[] = {
   0x00, 0xF8, 0x00, 0xFC, 0x00, 0xFE, 0x00, 0xFF, 0x80, 0xFF, 0xC0, 0x7F,
   0xE0, 0x3F, 0xF0, 0x1F, 0xF8, 0x0F, 0xFC, 0x07, 0xFC, 0x03, 0xFE, 0x01,
   0xFE, 0x00, 0x7F, 0x00, 0x1F, 0x00, 0x07, 0x00};
   .
   .
   .
/******* doCreateCursor *********/
static void doCreateCursor( )
{
    XColor dummy, cursor_foreground, cursor_background;

     /*Create pixmaps for cursor */
❶   pixmap = XCreatePixmap(dpy, XDefaultRootWindow(dpy), 1, 1, 1);

❷   XLookupColor(dpy, XDefaultColormapOfScreen(screen), "black",
        &dummy, &cursor_foreground);
    XLookupColor(dpy, XDefaultColormapOfScreen(screen), "white",
        &dummy, &cursor_background);
❸   pencil = XCreatePixmapFromBitmapData(dpy, pixmap, pencil_bits,
        pencil_width, pencil_height, 1, 0, 1);
    pencil_mask = XCreatePixmapFromBitmapData(dpy, pixmap, pencil_mask_bits,
        pencil_width, pencil_height, 1, 0, 1);

❹   pencil_cursor = XCreatePixmapCursor(dpy, pencil, pencil_mask,
        &cursor_foreground, &cursor_background, pencil_xhot, pencil_yhot);
    XDefineCursor(dpy, win, pencil_cursor);
}
```

❶ The client first creates a pixmap into which it will draw bit images for the cursor and cursor mask. Note that the depth of the pixmap must be one. For information about creating pixmaps, see Chapter 7.

❷ The LOOKUP COLOR routine returns the color value associated with the named color to the *cursor_foreground* and *cursor_background* variables. For information about LOOKUP COLOR, see Chapter 5.

❸ The CREATE PIXMAP FROM BITMAP DATA routine writes an image into a specified pixmap. The client uses the routine to write images for the cursor and the cursor mask into two pixmaps with depths of one.

❹ The CREATE PIXMAP CURSOR routine uses the two pixmaps to create the pixmap cursor.

6.8.2 Managing Cursors

To dissociate a cursor from a window, call the UNDEFINE CURSOR routine. After a call to UNDEFINE CURSOR, the cursor associated with the parent window is used. If the window is a root window, UNDEFINE CURSOR restores the default cursor. UNDEFINE CURSOR does not destroy a cursor. Using its identifier, the client can still refer to the cursor and associate it with a window.

To change the color of a cursor, use the RECOLOR CURSOR routine. If the cursor is displayed on the screen, the change is immediately visible. For information about defining foreground and background colors, see Chapter 5. For information about loading fonts, see Chapter 8.

6.8.3 Destroying Cursors

To destroy a cursor, use the FREE CURSOR routine. FREE CURSOR deletes the association between the cursor identifier and the specified cursor. It also frees memory allocated for the cursor.

7 Using Pixmaps and Images

Xlib enables clients to create and work with both on-screen graphics, such as lines and cursors, and off-screen images, such as pixmaps. Chapter 4 and Chapter 6 describe how to work with on-screen graphics objects.

This chapter describes how to work with off-screen graphics resources, including the following topics:

- Creating and freeing pixmaps
- Creating and managing bitmap files
- Working with images

7.1 Creating and Freeing Pixmaps

A **pixmap** is an area of memory into which clients can either define an image or temporarily save part of a screen. Pixmaps are useful for defining cursors and icons, for creating tiling patterns, and for saving portions of a window that has been exposed. Additionally, drawing complicated graphics sequences into pixmaps and then copying the pixmaps to a window is often faster than drawing the sequences directly to a window.

Use the CREATE PIXMAP routine to create a pixmap. The routine creates a pixmap of a specified width, height, and depth. If the width or height is zero or the depth is not supported by the drawable root window, the server returns an error. The pixmap must be associated with a window, which can be either an input-output or an input-only window.

Example 7–1 illustrates creating a pixmap to use as backing store for drawing the star of Example 6–5.

Example 7–1 Creating a Pixmap

```
    /* Create window win on                                    *
     *  display dpy, defined as follows:                       *
     *     Position: x = 100,y = 100                           *
     *     Width = 600                                         *
     *     Height = 600                                        *
     *  gc refers to the graphics context                      */

Pixmap pixmap;
int n, exposeflag = 0;
   .
   .
   .
/******** Create the graphics context *********/
static void doCreateGraphicsContext( )
{
    XGCValues xgcv;

    /* Create graphics context. */

❶   xgcv.foreground = doDefineColor(1);
    xgcv.background = doDefineColor(1);

    gc = XCreateGC(dpy, win, GCForeground | GCBackground, &xgcv);
}

/******* doCreatePixmap *********/
static void doCreatePixmap( )
{
    XPoint pt_arr[6];

    pt_arr[0].x = 75;
    pt_arr[0].y = 500;
    pt_arr[1].x = 300;
    pt_arr[1].y = 100;
    pt_arr[2].x = 525;
    pt_arr[2].y = 500;
    pt_arr[3].x = 50;
    pt_arr[3].y = 225;
    pt_arr[4].x = 575;
    pt_arr[4].y = 225;
    pt_arr[5].x = 75;
    pt_arr[5].y = 500;

❷   pixmap = XCreatePixmap(dpy, win, winW, winH, DefaultDepthOfScreen(screen));
❸   XFillRectangle(dpy, pixmap, gc, 0, 0, winW, winH);
    XSetForeground(dpy, gc, doDefineColor(2));
❹   XFillPolygon(dpy, pixmap, gc, &pt_arr, 6, Complex, CoordModeOrigin);
}
   .
   .
   .
/****************** doHandleEvents **********************/
static void doHandleEvents( )
{
    XEvent event;
```

(continued on next page)

Example 7–1 (Cont.) Creating a Pixmap

```
    for ( ; ; ) {
        XNextEvent(dpy, &event);
        switch (event.type) {
            case Expose:                doExpose(&event); break;
            case ButtonPress:           doButtonPress(&event); break;
        }
    }
}

/***** Write a message *****/
static void doExpose(eventP)
XEvent *eventP;
{
    char message1[ ] = {"To create a filled polygon, click MB1."};
    char message2[ ] = {"To exit, click MB2."};

    XDrawImageString(dpy, win, gc, 150, 25, message1, strlen(message1));
    XDrawImageString(dpy, win, gc, 150, 50, message2, strlen(message2));
❺   if (!exposeflag)
        exposeflag = 1;
    else
        XCopyArea(dpy, pixmap, win, gc, 0, 0, winW, winH, 0, 0);
        XDrawImageString(dpy, win, gc, 150, 50, message2, strlen(message2));
}

/***** Draw the polygon in the window *****/
static void doButtonPress(eventP)
XEvent *eventP;
{
    char message2[ ] = {"To exit, click MB2."};

    if (eventP->xbutton.button == Button2) sys$exit (1);

    XCopyArea(dpy, pixmap, win, gc, 0, 0, winW, winH, 0, 0);
    XDrawImageString(dpy, win, gc, 150, 50, message2, strlen(message2));
}
```

❶ Pixmaps use only the foreground member of the graphics context to define color. Because the client is using the pixmap as backing store, which is copied into the window to repaint exposed areas, both foreground and background members of the graphics context are first defined as the window background color.

❷ The pixmap has the width, height, and depth of the window.

❸ FILL RECTANGLE fills the pixmap with the background color of the window. After filling the pixmap to ensure that pixel values of both the pixmap and window background are the same, the foreground color is redefined for graphics operations.

❹ After redefining foreground color, the client draws the polygon into the pixmap. For description of specifying and filling the polygon, see Example 6–5.

❺ At the first window exposure, the client draws only the text into the window. On subsequent exposures, the client copies the pixmap into the window to repaint exposed areas. For a description of handling exposure events, see Chapter 9.

When a client no longer needs a pixmap, use the FREE PIXMAP routine to free storage associated with it. FREE PIXMAP first deletes the association between the pixmap identifier and the pixmap and then frees pixmap storage.

7.2 Creating and Managing Bitmaps

Xlib enables clients to create files of bitmap data and then use those files to create either bitmaps or pixmaps. To create a bitmap data file, use the WRITE BITMAP FILE routine. Example 7–2 illustrates creating a pixmap and writing the pixmap data into a bitmap data file.

Example 7–2 Creating a Bitmap Data File

```
        .
        .
        .
/******* doCreatePixmap *********/
static void doCreatePixmap( )
{
    XPoint pt_arr[5];

    pt_arr[0].x = 20;
    pt_arr[0].y = 0;
    pt_arr[1].x = 20;
    pt_arr[1].y = 5;
    pt_arr[2].x = 20;
    pt_arr[2].y = 10;
    pt_arr[3].x = 20;
    pt_arr[3].y = 15;
    pt_arr[4].x = 20;
    pt_arr[4].y = 20;

    pixmap = XCreatePixmap(dpy, win, pixW, pixH, DefaultDepthOfScreen(screen));
    XFillRectangle(dpy, pixmap, gc, 0, 0, pixW, pixH);
    XSetForeground(dpy, gc, doDefineColor(2));
    XDrawLines(dpy, pixmap, gc, &pt_arr, 5, CoordModeOrigin);
    status = XWriteBitmapFile(dpy, "bitfile.dat", pixmap, 20, 20, 0, 0);
}
```

The client first creates a pixmap using the method described in Section 7.1 and then calls the WRITE BITMAP FILE routine to write the pixmap data into the BITFILE.DAT bitmap file.

To create a bitmap or pixmap from a bitmap data file, use either the CREATE BITMAP FROM DATA or CREATE PIXMAP FROM DATA routine. Example 7–3 illustrates creating a pixmap from the bitmap data stored in BITFILE.DAT.

Example 7–3 Creating a Pixmap from Bitmap Data

```
   .
   .
   .
/******* doCreatePixmap *********/
static void doCreatePixmap( )
{
   static char LINES[] = {
      0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x3f, 0x06, 0x00,
      0x03, 0x0c, 0x00, 0x03, 0x18, 0x02, 0x03, 0x30, 0x00, 0xf3, 0x7f, 0x05,
      0x03, 0x30, 0x00, 0x03, 0x18, 0x00, 0x03, 0x0c, 0x00, 0x3f, 0x06, 0x00,
      0x00, 0x00, 0x05, 0x00, 0x00, 0x00, 0x00, 0x00, 0x04, 0x00, 0x00, 0x00,
      0xaa, 0xaa, 0x0a, 0x55, 0x55, 0x05, 0xaa, 0xaa, 0x0a, 0x55, 0x55, 0x05};

    pixmap = XCreatePixmapFromBitmapData(dpy, win, LINES, pixW, pixH,
        xgcv.foreground, xgcv.background, XDefaultDepthOfScreen(screen));
    XSetWindowBackgroundPixmap(dpy, win, pixmap);
}
   .
   .
   .
```

The client uses the pixmap to define window background.

7.3 Working with Images

Instead of managing images directly, clients perform operations on them by using the image data structure, which includes a pointer to data such as the LINES array defined in Example 7–3. In addition to the image data, the image data structure includes pointers to client-defined functions that perform the following operations:

- Destroying an image
- Getting a pixel from the image
- Storing a pixel in the image
- Extracting part of the image
- Adding a constant to the image

If the client has not defined a function, the corresponding Xlib routine is called by default.

The following illustrates the data structure:

```
typedef struct _XImage {
    int width, height;
    int xoffset;
    int format;
    char *data;
    int byte_order;
    int bitmap_unit;
    int bitmap_bit_order;
    int bitmap_pad;
    int depth;
    int bytes_per_line;
    int bits_per_pixel;
    unsigned long red_mask;
    unsigned long green_mask;
    unsigned long blue_mask;
    char *obdata;
    struct funcs {
        struct _XImage *(*create_image)();
        int (*destroy_image)();
        unsigned long (*get_pixel)();
        int (*put_pixel)();
        struct _XImage *(*sub_image)();
        int (*add_pixel)();
        } f;
} XImage;
```

Table 7–1 Image Data Structure Members

Member Name	Contents
width	Specifies the width of the image.
height	Specifies the height of the image.
offset	Specifies the number of pixels offset in the x direction. Specifying an offset permits the server to ignore the beginning of scanlines and rapidly display images when Z pixmap format is used.
format	Specifies whether the data is stored in XY pixmap or Z pixmap format. The following flags facilitate specifying data format:

Flag Name	Description
XYBitmap	A single bitmap representing one plane
XYPixmap	A set of bitmaps representing individual planes
ZPixmap	Data organized as a list of pixel values viewed as a horizontal row

Member Name	Contents
data	Address of the image data.
byte_order	Indicates whether least significant or most significant byte is first.

(continued on next page)

Table 7–1 (Cont.) Image Data Structure Members

Member Name	Contents
bitmap_unit	Specifies whether the bitmap is organized in units of 8, 16, or 32 bits.
bitmap_bit_order	Specifies whether the bitmap order is least or most significant.
bitmap_pad	Specifies whether padding in XY format or Z format should be done in units of 8, 16, or 32 bits.
depth	Specifies the depth of the image.
bytes_per_line	Specifies the bytes per line to be used as an accelerator.
bits_per_pixel	Indicates for Z format the number of bits per pixel.
red_mask	Specifies red values for Z format.
green_mask	Specifies green values for Z format.
blue_mask	Specifies blue values for Z format.
obdata	Address of a data structure that contains object routines.
create_image	Client-defined function that creates an image.
destroy_image	Client-defined function that destroys an image.
get_pixel	Client-defined function that gets the value of a pixel in the image.
put_pixel	Client-defined function that changes the value of a pixel in the image.
sub_image	Client-defined function that creates a new image from an existing one.
add_pixel	Client-defined function that increments the value of each pixel in the image by a constant.

To create an image, use either the CREATE IMAGE or the GET IMAGE routine. CREATE IMAGE initializes an image data structure, including a reference to the image data. For example, the following call creates an image data structure that points to the image data LINES, illustrated in Example 7–3:

```
#define pixW 16
#define pixH 16
#define bitmap_pad 16
#define bytes_per_line 16

XImage *image;
    .
    .
    .
     image = XCreateImage(dpy, XDefaultVisualOfScreen(screen),
          XDefaultDepthOfScreen(screen), ZPixmap, 0, &LINES,
          pixW, pixH, bitmap_pad, bytes_per_line);
    .
    .
    .
```

Note that the CREATE IMAGE routine does not allocate storage space for the image data.

To create an image from a drawable, use the GET IMAGE routine. In the following example, the client creates an image from a pixmap:

```
#define xOrigin 0
#define yOrigin 0
#define pixW 16
#define pixH 16
    .
    .
    .
    image = XGetImage(dpy, pixmap, xOrigin, yOrigin, pixW,
         pixH, AllPlanes, ZPixmap);
    .
    .
    .
```

To transfer an image from memory to a drawable, use the PUT IMAGE routine. In the following example, the client transfers the image from memory to a window:

```
#define pixW 16
#define pixH 16
#define srcX 0
#define srcY 0
#define dstX 200
#define dstY 200
    .
    .
    .
    XPutImage(dpy, win, gc, image, srcX, srcY, dstX, dstY,
        pixW, pixH);
    .
    .
    .
```

The call transfers the entire image, which was created in the call to GET IMAGE, from memory to coordinates (200, 200) in the window.

As the description of the image data structure indicates, Xlib enables clients to store an image in the following ways:

- As a bitmap—XY bitmap format stores the image as a two-dimensional array. Figure 7–1 illustrates XY bitmap format.
- As a set of bitmaps—XY pixmap format stores the image as a stack of bitmaps. Figure 7–2 illustrates XY pixmap format.
- As a list of pixel values—Z pixmap format stores the image as a list of pixel values viewed as a horizontal row. Each example of creating an image uses Z pixmap format. Figure 7–3 illustrates scanline order.

Figure 7–1 XY Bitmap Format

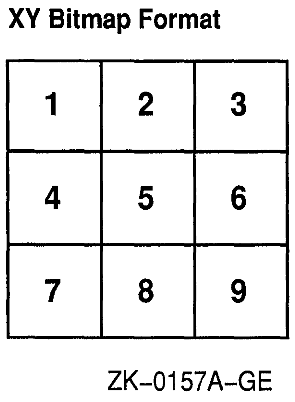


Figure 7–2 XY Pixmap Format

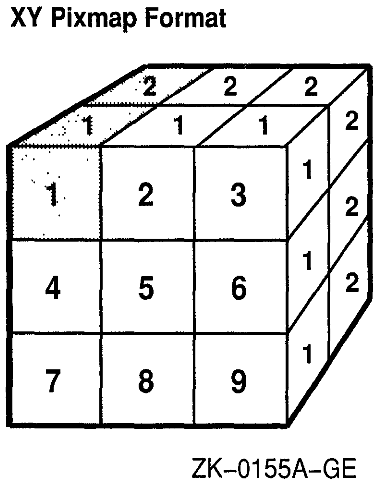
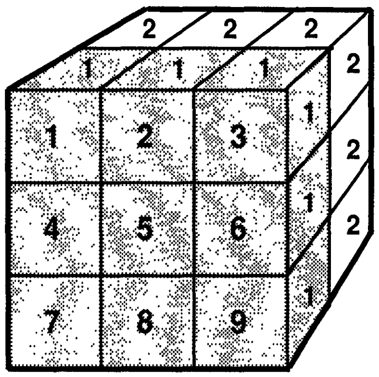


Figure 7–3 Z Format

Z Pixmap Format



ZK–0156A–GE

Xlib includes routines to change images by manipulating their pixel values and creating new images out of subsections of existing images. Table 7–2 lists these routines and their use. Clients can override these routines by defining functions referred to in the image data structure.

Table 7–2 Routines That Change Images

Routine	Description
ADD PIXEL	Increments each pixel in an image by a constant value
GET PIXEL	Returns the pixel value of an image
PUT PIXEL	Sets the pixel value of an image
SUB IMAGE	Creates a new image out of a subsection of an existing image

When a client no longer needs an image, use the DESTROY IMAGE routine to deallocate memory associated with the image data structure.

8 Writing Text

This chapter describes writing text using Xlib. The chapter includes the following topics:

- Characters and fonts—A description of the composition of characters and types of fonts and their components
- Specifying fonts—How to load a font and associate it with a graphics context
- Computing text size—How to determine the size of text
- Getting information about text—How to get information about text
- Drawing text—How to write text on the screen

VMS DECwindows provides a font compiler to enable programmers to convert ASCII files into binary form. For a guide to using the font compiler, see Appendix A.

8.1 Characters and Fonts

The smallest unit of text the server displays on a screen is a **character**. Pixels that form a character are enclosed within a **bounding box** that defines the number of pixels the server turns on or off to represent the character on the screen. For example, Figure 8–1 illustrates the bounding box that encloses the character "y."

The server turns each pixel within the bounding box either on or off, depending on the character. Consequently, bounding box size affects performance. Larger bounding boxes require more server time to process than do smaller boxes.

The character is positioned relative to the **baseline** and the character origin. The baseline is logically viewed as the x axis that runs just below nondescending characters. The **character origin** is a point along the baseline. The **left bearing** of the character is the distance from the origin to the left edge of the bounding box; the **right bearing** is the distance from the origin to the right edge. **Ascent** and **descent** measure the distance from the baseline to the top and bottom of the bounding box, respectively. **Character width** is the distance from the origin to the next character origin ($x + width, y$).

Figure 8–1 Composition of a Character

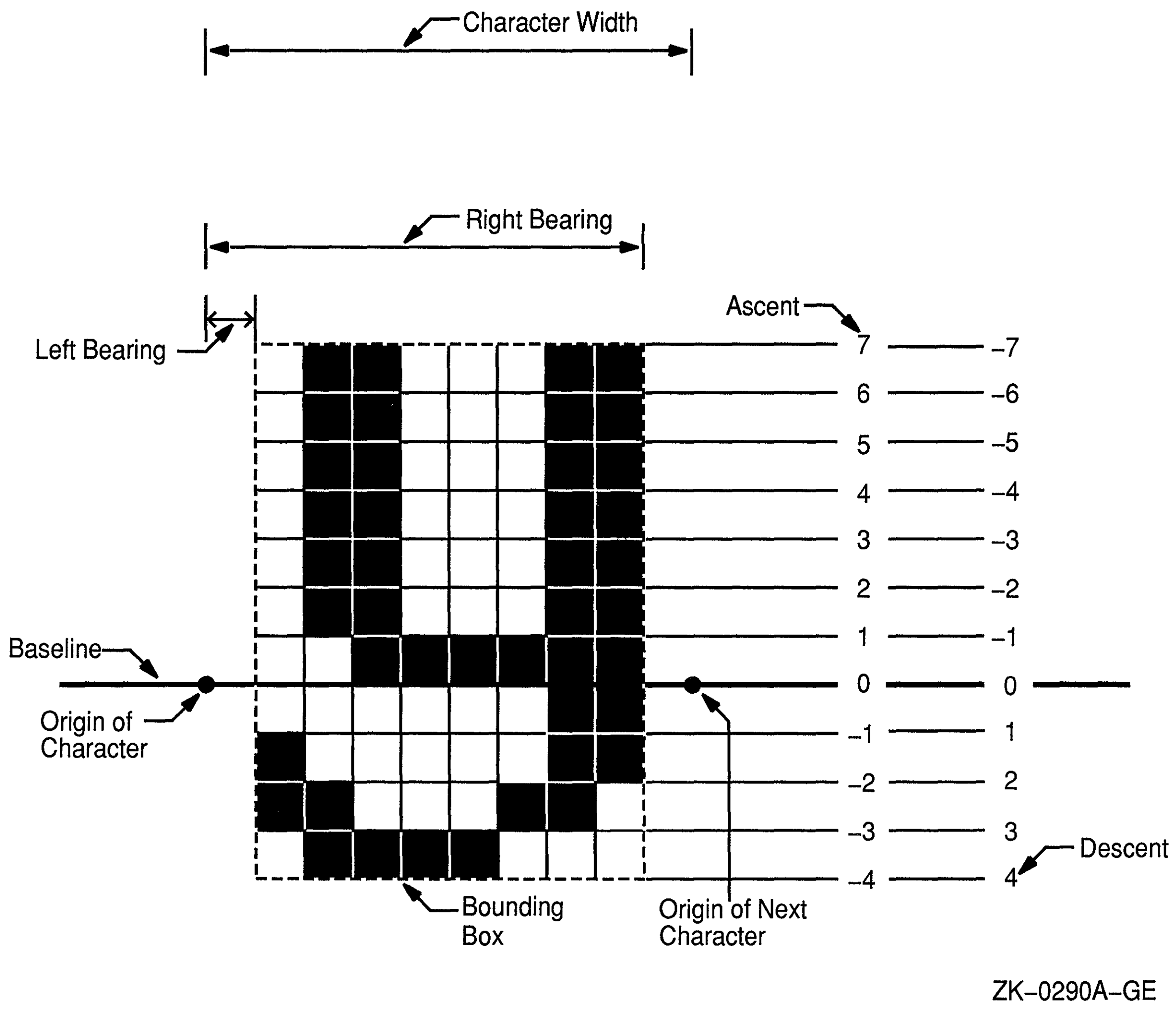
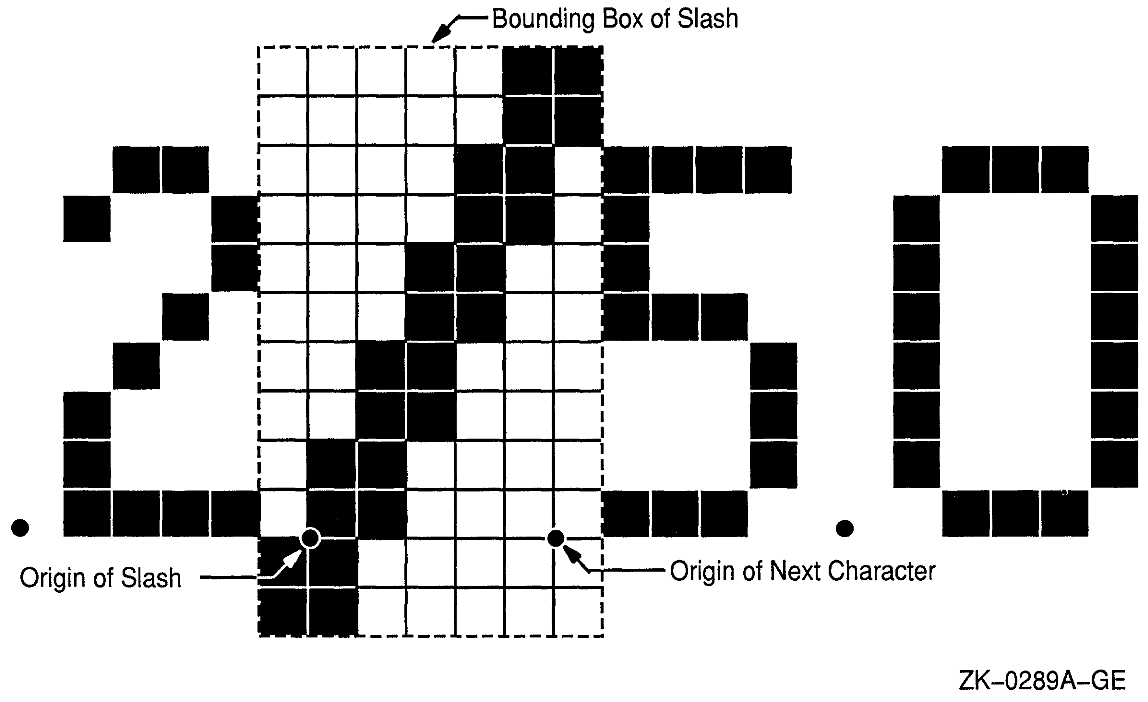


Figure 8–2 illustrates that the bounding box of a character can extend beyond the character origin. The bounding box of the back slash extends one pixel to the left of the origin of the slash, giving the character a left bearing of −1. The back slash is also unusual because its bounding box extends to the right of the next character. The width of the slash, measured from origin to origin, is 5; the right bearing, measured from origin to the right edge of the bounding box, is 6.

Figure 8–2 Composition of a Back Slash



The left bearing, right bearing, ascent, descent, and width of a character are **character metrics**. Xlib maintains information about character metrics in a char struct data structure. The following illustrates the data structure:

```
typedef struct {
    short       lbearing;
    short       rbearing;
    short       width;
    short       ascent;
    short       descent;
    unsigned short attributes;
} XCharStruct;
```

Table 8–1 describes members of the char struct data structure. Any member of the data structure can have a negative value, except the attributes member.

Table 8–1 Char Struct Data Structure Members

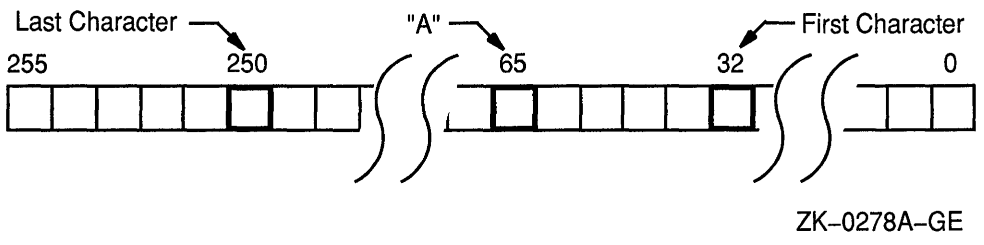
Member Name	Contents
lbearing	Distance from the origin to the left edge of the bounding box. When the value of this member is zero, the server draws only pixels whose x-coordinates are less than the value of the origin x-coordinate.
rbearing	Distance from the origin to the right edge of the bounding box.
width	Distance from the current origin to the origin of the next character. Text written left to right, such as Arabic, uses a negative width to place the next character to the left of the current origin.
ascent	Distance from the baseline to the top of the bounding box.
descent	Distance from the baseline to the bottom of the bounding box.
attributes	Attributes of the character defined in the bitmap distribution format (BDF) file. A character is not guaranteed to have any attributes.

A **font** is a group of characters that have the same style and size. Xlib supports both fixed and proportional fonts. A **fixed font** has equal metrics. For example, all characters in the font have the same value for left bearing. Consequently, the bounding box for all characters is the same. All metrics in a **proportional font** can vary from character to character. A **monospaced font** is a special type of proportional font in which only the width of all characters must be equal. Bounding boxes of characters in a monospaced font vary depending on the size of characters. If the same font is compiled as a monospaced font and a fixed font, the bounding boxes of the monospaced font are typically smaller than the bounding box that encloses fixed-font characters. For information about compiling fonts, see Appendix A.

Xlib uses indexes to refer to characters that compose a font. The indexes, each defined by a byte, are arranged in one or more rows of up to 256 indexes. A font can contain as many as 256 rows of character indexes, used contiguously. Fonts seldom use all possible indexes.

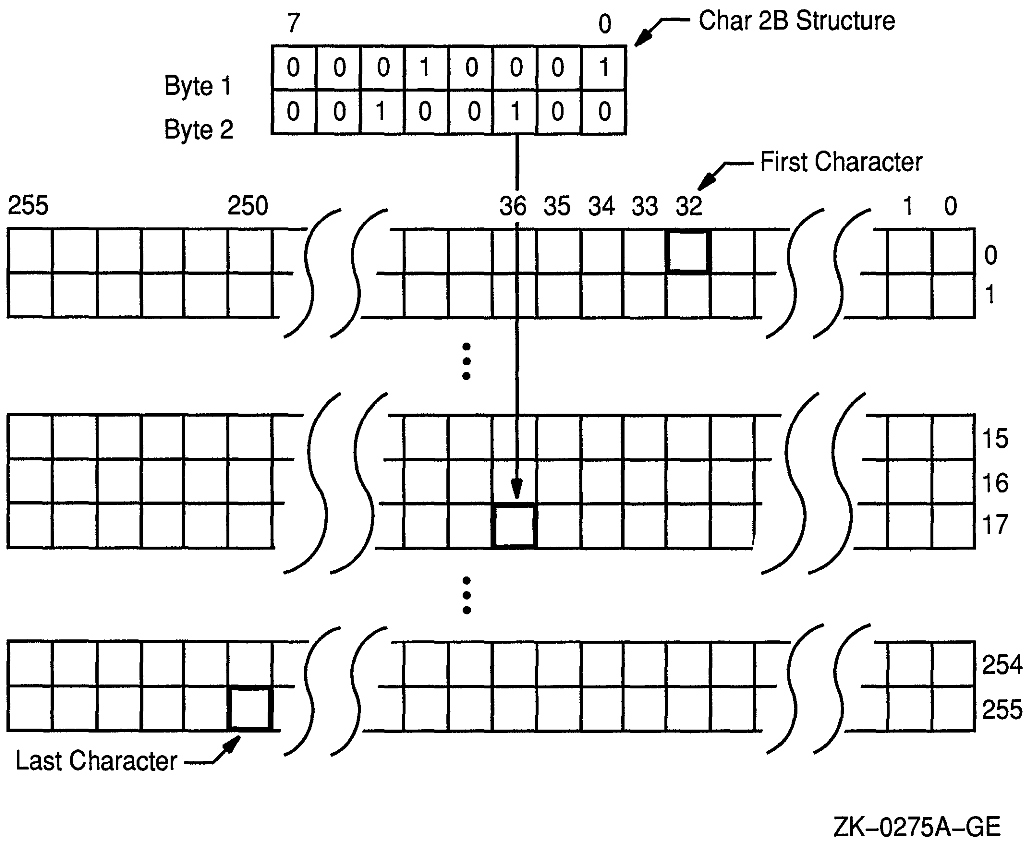
For example, the font illustrated in Figure 8–3, comprises 219 characters in columns 32 through 250, one column for each character index. Columns 0 through 31 and 251 through 255 are undefined. The first character of the font is located at column 32; the last character is located at column 250. Because all characters are defined in one row of 256 indexes, the font is a **single-row font**. In the illustration, character "A" is located at column 65.

Figure 8–3 Single-Row Font



Multiple-row fonts, such as Kanji, comprise more characters than can be indexed by a single row of 256 bytes. Figure 8–4 illustrates the configuration of a multiple-row font. Byte 1 refers to the row. Byte 2 refers to the column in the row. In Figure 8–4, the character is located at column 36 in row 17. Note that each row of a multiple-row font has the same number of undefined bytes at the beginning and end. In each row, characters begin at column 32 and end at column 250.

Figure 8–4 Multiple-Row Font



Xlib provides a char 2B data structure to enable clients to index multiple-row fonts easily. The following illustrates the data structure:

```
typedef struct {
    unsigned char byte1;
    unsigned char byte2;
} XChar2b;
```

Table 8–2 describes members of the data structure.

Table 8–2 Char 2B Data Structure Members

Member Name	Contents
byte1	Row in which the character is indexed
byte2	Position of the character in the row

Xlib maintains a record of the characteristics of a font in the font struct data structure. The following illustrates the font struct data structure:

```
typedef struct {
    XExtData    *ext_data;
    Font        fid;
    unsigned    direction;
    unsigned    min_char_or_byte2;
    unsigned    max_char_or_byte2;
    unsigned    min_byte1;
    unsigned    max_byte1;
    Bool        all_chars_exist;
    unsigned    default_char;
    int         n_properties;
    XFontProp   *properties;
    XCharStruct min_bounds;
    XCharStruct max_bounds;
    XCharStruct *per_char;
    int         ascent;
    int         descent;
} XFontStruct;
```

Table 8–3 describes members of the data structure.

Table 8–3 Font Struct Data Structure Members

Member Name	Contents
ext_data	Data used by extensions.
fid	Identifier of the font.
direction	Hint about the direction in which the font is painted. The direction can be either left to right, specified by the constant FontLeftToRight, or right to left, specified by the constant FontRightToLeft. The core protocol does not support vertical text.
min_char_or_byte2	First character in the font.

(continued on next page)

Table 8–3 (Cont.) Font Struct Data Structure Members

Member Name	Contents
max_char_or_byte2	Last character in the font.
min_byte1	First row that exists.
max_byte1	Last row that exists.
all_chars_exist	If the value of this member is true, all characters in the array pointed to by per_char have nonzero bounding boxes.
default_char	Character used when an undefined or nonexistent character is printed. The default character is a 16-bit, not a 2-byte, character. For a multiple-row font, default_char has byte 1 in the most significant byte and byte 2 in the least significant byte. If default_char specifies an undefined or nonexistent character, the server does not print an undefined or nonexistent character.
n_properties	Number of properties associated with the font.
properties	Address of an array of additional font properties.
min_bounds	The minimum bounding box value of all the elements in the array of char struct data structures that define each character in the font. For a description of the use of min_bounds see max_bounds.

max_bounds	Maximum metrics values of all the characters in the font.
	 Using the values of min_bounds and max_bounds, clients can compute the bounding box of the font. The bounding box of the font is determined by first computing the minimum and maximum values of the left bearing, right bearing, width, ascent, and descent of all characters and then subtracting minimum from maximum values. The upper left coordinate of the bounding box (x,y) is defined as follows:
	 `x + min_bounds.lbearing, y - max_bounds.ascent`
	 The width of the font bounding box is defined as follows:
	 `max_bounds.rbearing - min_bounds.lbearing`
	 Note that this is not the width of a character in the font.
	 The height is defined as follows:
	 `max_bounds.ascent + max_bounds.descent`

per_char	Address of an array of char struct data structures that define each character in the font.
ascent	Distance from the baseline to the top of the bounding box. With descent, ascent is used to determine line spacing. Specific characters in the font may extend beyond the font ascent.

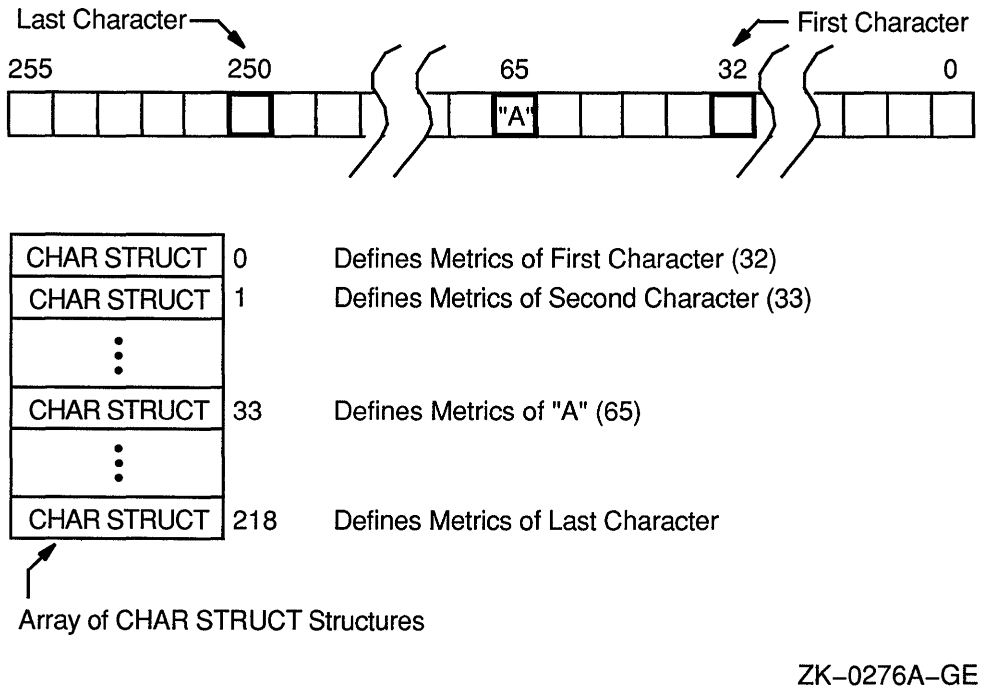
(continued on next page)

Table 8–3 (Cont.) Font Struct Data Structure Members

Member Name	Contents
descent	Distance from the baseline to the bottom of the bounding box. With ascent, descent is used to determine line spacing. Specific characters in the font may extend beyond the font descent.

As Table 8–3 indicates, Xlib records metrics for each character in an array of char struct data structures specified by the font struct per_char member. The array comprises as many char struct data structures as there are characters in the font. However, the indexes that refer to the location of characters in the array differ from the indexes to characters in the font. For example, 32 indexes the first character of the font illustrated in Figure 8–5, whereas 0 indexes its char struct data structure in the array.

Figure 8–5 Indexing Single-Row Font Character Metrics


Last Character
First Character
255
250
65
32
0
"A"
CHAR STRUCT 0 Defines Metrics of First Character (32)
CHAR STRUCT 1 Defines Metrics of Second Character (33)
CHAR STRUCT 33 Defines Metrics of "A" (65)
CHAR STRUCT 218 Defines Metrics of Last Character
Array of CHAR STRUCT Structures
ZK–0276A–GE

To locate the char struct data structure that defines the metrics of any character in a single-row font, subtract the value of the column that indexes the first character in the font, specified by min_char_or_byte2, from the position of the character. For instance, in Figure 8–5 the metrics of character “A” are located at index 33 in the array of char struct data structures specified by the per_char member.

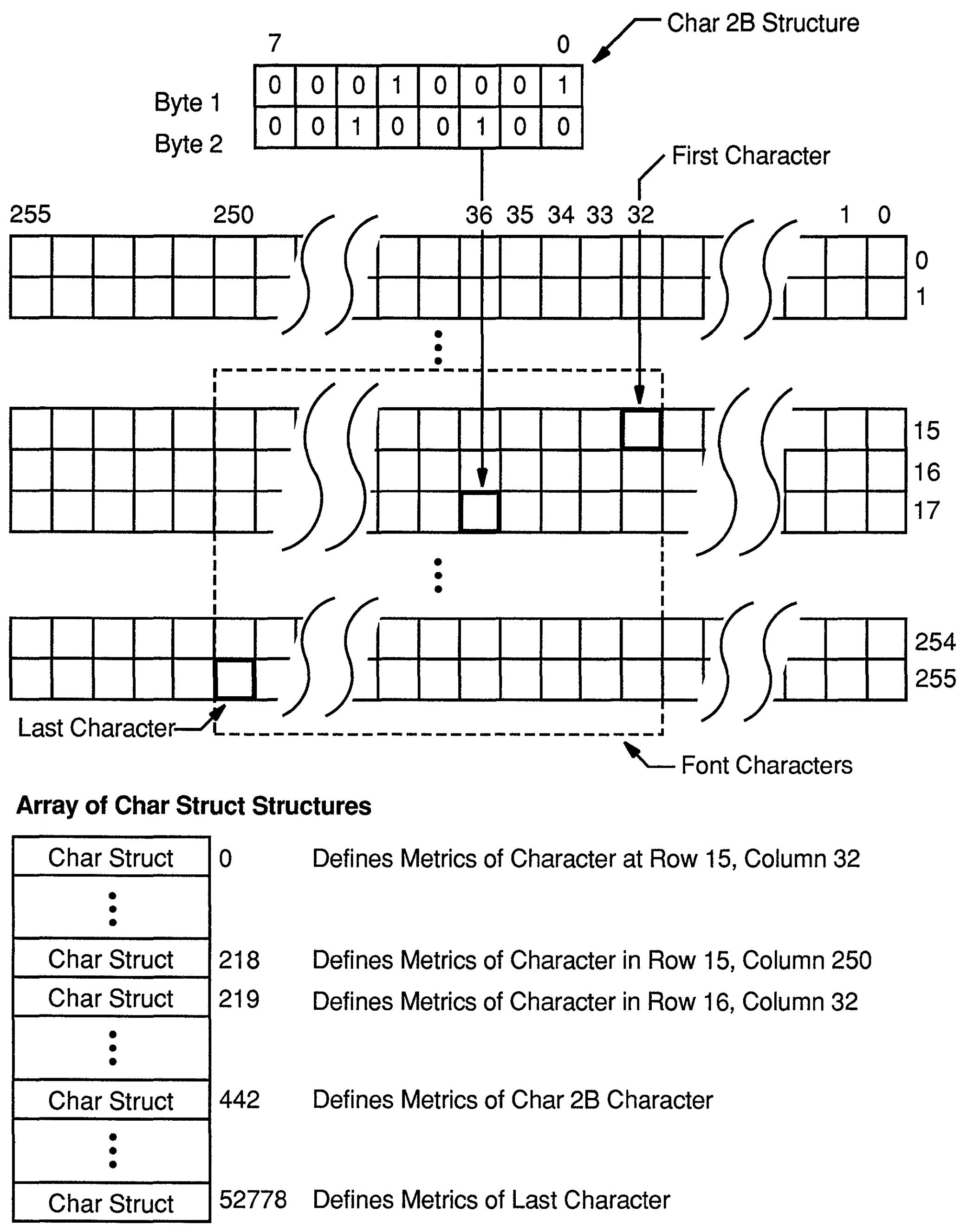
To locate the char struct data structure that defines the metrics of a character of a multiple-row font, use the following formula to adjust for both the number of rows in the font and the position of the character in a row:

$$(row - first\ row\ of\ characters) * N + (position\ in\ column - first\ column)$$

N is equal to the last column minus the first column plus 1.

For example, the array index of the character specified in Figure 8–6 is 442.

Figure 8–6 Indexing Multiple-Row Font Character Metrics


Char 2B Structure
7
0
Byte 1
0 0 0 1 0 0 0 1
Byte 2
0 0 1 0 0 1 0 0
First Character
255
250
36 35 34 33 32
1 0
0
1
15
16
17
254
255
Last Character
Font Characters
Array of Char Struct Structures
Char Struct 0 Defines Metrics of Character at Row 15, Column 32
Char Struct 218 Defines Metrics of Character in Row 15, Column 250
Char Struct 219 Defines Metrics of Character in Row 16, Column 32
Char Struct 442 Defines Metrics of Char 2B Character
Char Struct 52778 Defines Metrics of Last Character
ZK-0277A-GE

Like windows, fonts may have properties associated with them. However, font properties differ from window properties. Window properties are data associated with windows; font properties describe font characteristics, such as spacing between words. When the font is compiled, its properties are defined in an array of font prop data structures.

Just as atoms name window properties, atoms name font properties. If the atoms are predefined, they have associated literals. For example, the predefined atom that identifies the height of capitalized letters is referred to by the literal XA_CAP_HEIGHT.

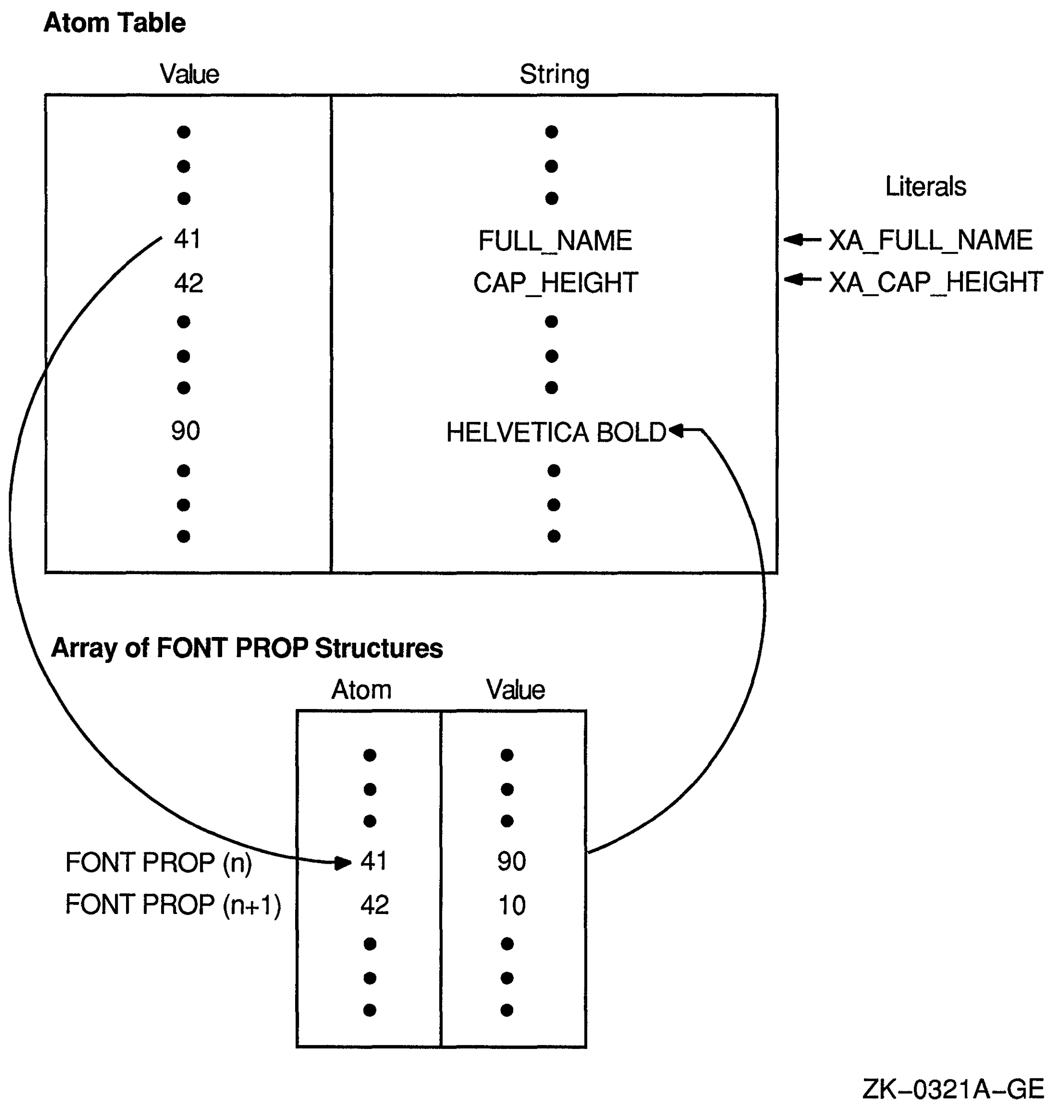
When working with properties, clients must know beforehand how to interpret the font property identified by an atom. Figure 8–7 illustrates this concept.

The server maintains an atom table for font properties. The table associates values with strings. For example, the atom table illustrated in Figure 8–7 defines two atoms. One associates the string FULL_NAME with the value 41. The other associates the string CAP_HEIGHT with the value 42. Notice that the string in the atom table is different from XA_FULL_NAME, the literal that refers to the atom.

Both atoms uniquely identify different types of data. FULL_NAME identifies string data that names the font. CAP_HEIGHT identifies integer data that defines the height of capitalized letters.

Although the atoms identify different types of data, the property table illustrated in Figure 8–7 associates both atoms with integers. The integer associated with CAP_HEIGHT defines without further interpretation the height of capitalized letters. However, the integer listed with FULL_NAME is an atom value. This integer, 90, corresponds to a value in the atom table that has an associated string, HELVETICA BOLD. To use the string, the client must know that the value associated with the atom is itself an atom value.

Figure 8–7 Atoms and Font Properties



Xlib lists each font property and its corresponding atom in a font prop data structure. The property value table in Figure 8–7 is an array of font prop data structures.

The following illustrates the data structure:

```
typedef struct {
    Atom name;
    unsigned long card32;
} XFontProp;
```

Table 8–4 describes members of the data structure.

Table 8–4 Font Prop Data Structure Members

Member Name	Contents
name	String of characters that names the property
card32	A 32-bit value that defines the font property

8.2 Specifying a Font

To specify a font for writing text, first load the font and then associate the loaded font with a graphics context. Appendix D lists VMS DECwindows fonts.

To load a font, use either the LOAD FONT or the LOAD QUERY FONT routine. LOAD FONT loads the specified font and returns a font identifier. LOAD QUERY FONT loads the specified font and returns information about the font to a font struct data structure.

Because LOAD QUERY FONT returns information to a font struct data structure, calling the routine takes significantly longer than calling LOAD FONT, which returns only the font identifier.

When using either routine, pass the display identifier and font name. Xlib font names consist of the following fields, in left to right order:

1. Foundry that supplied the font, or the font designer
2. Typeface family of the font
3. Weight (book, demi, medium, bold, light)
4. Style (R (roman), I (italic), O (oblique))
5. Width per horizontal unit of the font (normal, wide, double wide, narrow)
6. Additional style font identifier
7. Pixel font size
8. Point size (8, 10, 12, 14, 18, 24)
9. Resolution in pixels/dots per inch
10. Spacing (monospaced, proportional, or character cell)
11. Average width of all characters in the font
12. Set character encoding

The full name of a representative font in SYS$SYSROOT:[DECW$FONT.100DPI] is as follows:

```
-ADOBE-ITC Avant Garde Gothic-Book-R-Normal--14-100-100-100-P-80-ISO8859-1
```

The font is named ITC Avant Garde Gothic. Font weight is book, font style is R (roman), and width between font units is normal.

The pixel size is 14 and the decipoint size is 100.

Horizontal and vertical resolution in dots per inch (dpi) is 100. When the dpi is 100, 14 pixels are required to be a 10 point font.

The font is proportionally spaced. Average width of characters is 80. Character encoding is ISOLATIN1.

The following designates the full name of the comparable font designed for a 75 dpi system:

```
-ADOBE-ITC Avant Garde Gothic-Book-R-Normal--10-100-75-75-P-59-ISO8859-1
```

Unlike the previous font, this font requires only 10 pixels to be 10 points. Note that this font differs from the previous font only in pixel size, resolution, and character width.

Xlib enables clients to substitute a question mark for a single character and an asterisk for one or more fields in a font name. The following illustrates using the asterisk to specify a 10-point ITC Avant Garde Gothic font of book weight, roman style, and normal spacing for display on either 75 or 100 dpi systems:

```
-ADOBE-ITC Avant Garde Gothic-Book-R-Normal--*-100-*-*-P-*
```

When using the asterisk, make sure that substitutions are clearly defined. For example, the following name ambiguously specifies two fonts:

```
-ADOBE-ITC Avant Garde Gothic-Book-R-Normal--*-100-*-P-*
```

Because the leftmost asterisk substitutes for all fields before the 100, the name defines the following two 100 dpi fonts:

```
-ADOBE-ITC Avant Garde Gothic-Book-R-Normal--11-80-100-100-P-80-ISO8859-1
-ADOBE-ITC Avant Garde Gothic-Book-R-Normal--14-100-100-100-P-80-ISO8859-1
```

The first is an 8 point font. The second is a 10 point font.

The following example illustrates loading the 10-point font:

```
#define FontName "-ADOBE-ITC Avant Garde
          Gothic-Book-R-Normal--*-100-*-*-P-*  "
.
.
.
font = XLoadFont(dpy, FontName);
.
.
.
```

After loading a font, associate it with a graphics context by calling the SET FONT routine. Specify the font identifier that either LOAD FONT or LOAD QUERY FONT returned, and a graphics context, as in the following example:

```
XSetFont(dpy, gc, font);
```

The call associates *font* with *gc*.

8.3 Getting Information About a Font

Xlib provides clients with routines that list available fonts, get font information with or without character metrics, and return the value of a specified font property.

To get a list of available fonts, use the LIST FONTS routine, specifying the font searched for.

LIST FONTS returns a list of available fonts that match the specified font name. When the client no longer needs the list of font names, call the FREE FONT NAMES routine to free storage allocated for the font list.

To receive both a list of fonts and information about the fonts, use the LIST FONTS WITH INFO routine. LIST FONTS WITH INFO returns both a list of fonts that match the font specified by the client and the address of a font struct data structure for each font listed. Each data structure contains information about the font. The data structure does not include character metrics in the per_char member. For a description of the information returned, see Table 8–3.

To receive information about a font, including character metrics, use the QUERY FONT routine. Because the server returns character metrics, calling QUERY FONT takes approximately eight times longer than calling LIST FONTS WITH INFO. To get the value of a specified property, use the GET FONT PROPERTY routine.

Although a font is not guaranteed to have any properties, it should have at least the properties described in Table 8–5. The table lists properties by atom name and data type. For information about properties, see Section 3.5.

Table 8–5 Atom Names of Font Properties

Atom	Data Type	Description of the Property
XA_MIN_SPACE	Unsigned	Minimum interword spacing, in pixels.
XA_NORMAL_SPACE	Unsigned	Normal interword spacing, in pixels.
XA_MAX_SPACE	Unsigned	Maximum interword spacing, in pixels.
XA_END_SPACE	Unsigned	Additional spacing at the end of a sentence, in pixels.
XA_SUPERSCRIPT_X	Signed	With XA_SUPERSCRIPT_Y, the offset from the character origin where superscripts should begin, in pixels. If the origin is [x, y], superscripts should begin at the following coordinates: `x + XA_SUPERSCRIPT_X, y - XA_SUPERSCRIPT_Y`
XA_SUPERSCRIPT_Y	Signed	With XA_SUPERSCRIPT_X, the offset from the character origin where superscripts should begin, in pixels. See the description under XA_SUPERSCRIPT_X.

(continued on next page)

Table 8–5 (Cont.) Atom Names of Font Properties

Atom	Data Type	Description of the Property
XA_SUBSCRIPT_X	Signed	With XA_SUBSCRIPT_Y, the offset from the character origin where subscripts should begin, in pixels. If the origin is [x, y], subscripts should begin at the following coordinates: `x + XA_SUBSCRIPT_X, y + XA_SUBSCRIPT_Y`
XA_SUBSCRIPT_Y	Signed	With XA_SUBSCRIPT_X, the offset from the character origin where subscripts should begin, in pixels. See the description under XA_SUBSCRIPT_X.
XA_UNDERLINE_POSITION	Signed	The y offset from the baseline to the top of an underline, in pixels. If the baseline y-coordinate is y, then the top of the underline is at y + XA_UNDERLINE_POSITION.
XA_UNDERLINE_THICKNESS	Unsigned	Thickness of the underline, in pixels.
XA_STRIKEOUT_ASCENT	Signed	With XA_STRIKEOUT_DESCENT, the vertical extent for boxing or voiding characters, in pixels. If the baseline y-coordinate is y, the top of the strikeout box is y - XA_STRIKEOUT_ASCENT. The height of the box is as follows: `XA_STRIKEOUT_ASCENT + XA_STRIKEOUT_DESCENT`
XA_STRIKEOUT_DESCENT	Signed	With XA_STRIKEOUT_ASCENT, the vertical extent for boxing or voiding characters, in pixels. See the description under XA_STRIKEOUT_ASCENT.
XA_ITALIC_ANGLE	Signed	The angle of the cominant staffs of characters in the font, in degrees scaled by 64, relative to the 3-o'clock position from the character origin. Positive values indicate counterclockwise motion.
XA_X_HEIGHT	Signed	One ex, as in TeX, but expressed in units of pixels. Often the height of lowercase x.
XA_QUAD_WIDTH	Signed	One em, as in TeX, but expressed in units of pixels. Often the width of the digits 0 to 9.
XA_CAP_HEIGHT	Signed	The y offset from the baseline to the top of capital letters, ignoring ascents. If the baseline y-coordinate is y, the top of the capitals is at y - XA_CAP_HEIGHT.
XA_WEIGHT	Unsigned	Weight or boldness of the font, expressed as a value between 0 and 1000.
XA_POINT_SIZE	Unsigned	Point size of the font at ideal resolution, expressed in 1/10 points.
XA_RESOLUTION	Unsigned	Number of pixels per point, expressed in 1/100, at which the font was created.
XA_COPYRIGHT	Unsigned	Copyright date of the font.
XA_NOTICE	Unsigned	Copyright date of the font name.

(continued on next page)

Table 8–5 (Cont.) Atom Names of Font Properties

Atom	Data Type	Description of the Property
XA_FONT_NAME	Atom	Font name.
XA_FAMILY_NAME	Atom	Name of the font family.
XA_FULL_NAME	Atom	Full name of the font.

8.4 Computing the Size of Text

Use the TEXT WIDTH and TEXT WIDTH 16 routines to compute the width of 8-bit and 2-byte strings, respectively. The routines return the sum of the width of each character in the specified string. To compute the bounding box of a specified 8-bit string, use either the TEXT EXTENTS or QUERY TEXT EXTENTS routine. Both TEXT EXTENTS and QUERY TEXT EXTENTS return the direction hint, ascent, descent, and overall size of the character string being queried.

TEXT EXTENTS passes to Xlib the font struct data structure returned by a previous call to either LOAD QUERY FONT or QUERY FONT. QUERY TEXT EXTENTS queries the server for font information, which the server returns to a font struct data structure. Because Xlib can process TEXT EXTENTS locally, without querying the server for font metrics, calling TEXT EXTENTS is significantly faster than calling QUERY TEXT EXTENTS.

To compute the bounding boxes of a specified 2-byte string, use either the TEXT EXTENTS 16 or the QUERY TEXT EXTENTS 16 routine. Both routines return information identical to information returned by TEXT EXTENTS and QUERY TEXT EXTENTS. As with TEXT EXTENTS, calling TEXT EXTENTS 16 is significantly faster than calling QUERY TEXT EXTENTS 16 because Xlib can process the call without making the round-trip to the server.

8.5 Drawing Text

Xlib enables clients to draw text stored in text data structures, text whose foreground bits are only displayed, and text whose foreground and background bits are displayed.

To draw 8-bit or 2-byte text stored in data structures, use either the DRAW TEXT or the DRAW TEXT 16 routine. Xlib includes text item and text item 16 data structures to enable clients to store text. The following illustrates the text item data structure:

```
typedef struct {
    char *chars;
    int nchars;
    int delta;
    Font font;
} XTextItem;
```

Table 8–6 describes members of the text item data structure.

Table 8–6 Text Item Data Structure Members

Member Name	Contents
chars	Address of a string of characters.
nchars	Number of characters in the string.
delta	Horizontal spacing before the start of the string. Spacing is always added to the string origin and is not dependent on the font used.
font	Identifier of the font used to print the string. If the value of this member is None, the server uses the current font in the GC data structure. If the member has a value other than None, the specified font is stored in the GC data structure.

The following illustrates the text item 16 data structure:

```
typedef struct {
    XChar2b *chars;
    int nchars;
    int delta;
    Font font;
} XTextItem16;
```

Table 8–7 describes members of the text item 16 data structure.

Table 8–7 Text Item 16 Data Structure Members

Member Name	Contents
chars	Address of a string of characters stored in a char 2B data structure. For a description of the char 2B data structure, see the description immediately following this table.
nchars	Number of characters in the string.
delta	Horizontal spacing before the start of the string. Spacing is always added to the string origin and is not dependent on the font used.
font	Identifier of the font used to print the string. If the value of this member is None, the server uses the current font in the GC data structure. If the member has a value other than None, the specified font is stored in the GC data structure.

Xlib provides a char 2B data structure to enable clients to store 2-byte text. The following illustrates the data structure:

```
typedef struct {
    unsigned char byte1;
    unsigned char byte2;
} XChar2b;
```

Xlib processes each text item in turn. Each character image, as defined by the font in the graphics context, is treated as an additional mask for a fill operation on the drawable. The drawable is modified only where the font character has a bit set to 1.

Example 8–1 illustrates using the DRAW TEXT routine to draw three words in one call.

Example 8–1 Drawing Text Using the DRAW TEXT Routine

```
   .
   .
   .
#define FirstFont "-ADOBE-NEW CENTURY SCHOOLBOOK-BOLD-R-NORMAL--*-80-*-*-P-*"
#define SecondFont "-ADOBE-NEW CENTURY SCHOOLBOOK-BOLD-R-NORMAL--*-140-*-*-P-*"
#define ThirdFont "-ADOBE-NEW CENTURY SCHOOLBOOK-BOLD-R-NORMAL--*-240-*-*-P-*"

Display *dpy;
Window window;
GC gc;
Screen *screen;
int n;
XTextItem text[] = {
        "small", 5, 0, 0,
        "bigger", 6, 0, 0,
        "biggest", 7, 0, 0,
};
   .
   .
   .
/******* Load the font for text writing ******/
static void doLoadFont( )
{
    Font FontOne, FontTwo, FontThree;

    FontOne = XLoadFont(dpy, FirstFont);
    FontTwo = XLoadFont(dpy, SecondFont);
    FontThree = XLoadFont(dpy, ThirdFont);
    XSetFont(dpy, gc, FontTwo);

    text[0].delta = 0;
    text[0].font = FontOne;

    text[1].delta = 20;
    text[1].font = FontTwo;

    text[2].delta = 20;
    text[2].font = FontThree;
}
   .
   .
   .
/**************** doButtonPress **************************/
static void doButtonPress(eventP)
XEvent *eventP;
{
    if (eventP->xbutton.button == Button2) sys$exit (1);
    if (eventP->xbutton.button == Button1)
        XDrawText(dpy, window, gc, 100, 300, text, 3);
}
```

To draw 8-bit or 2-byte text, use the DRAW STRING, DRAW STRING 16, DRAW IMAGE STRING, and DRAW IMAGE STRING 16 routines. DRAW STRING and DRAW STRING 16 display the foreground values of text only. DRAW IMAGE STRING and DRAW IMAGE STRING 16 display both foreground and background values.

Example 8–2 illustrates drawing text with the DRAW STRING routine. The example modifies the sample program in Chapter 1 to draw shadow text.

Example 8–2 Drawing Text Using the DRAW STRING Routine

```
    .
    .
    .
/***** Write the message in the window *****/
static void doExpose(eventP)
XEvent *eventP;
{
    if (eventP->xexpose.window != window2) return;
    XClearWindow(dpy, window2);
    XSetForeground(dpy, gc, doDefineColor(3));
    XDrawString(dpy, window2, gc, 35, 75, message[state], strlen(message[state]));
    XSetForeground(dpy, gc, doDefineColor(4));
    XDrawString(dpy, window2, gc, 31, 71, message[state], strlen(message[state]));
}

/***************** doShutdown ***************************/
static void doButtonPress(eventP)
XEvent *eventP;
{
    if (eventP->xexpose.window != window2) {
    state = 1;
    XClearWindow(dpy, window2);
    XSetForeground(dpy, gc, doDefineColor(3));
    XDrawString(dpy, window2, gc, 35, 75, message[state], strlen(message[state]));
    XSetForeground(dpy, gc, doDefineColor(4));
    XDrawString(dpy, window2, gc, 31, 71, message[state], strlen(message[state]));
    return;
    }

    /* Unmap and destroy windows */

    XUnmapWindow(dpy, window1);
    XDestroyWindow(dpy, window1);

    XCloseDisplay(dpy);

    sys$exit (1);
}
```

The server refers to the following members of the GC data structure when writing text with DRAW TEXT, DRAW TEXT 16, DRAW STRING, and DRAW STRING 16:

Function	Plane mask
Foreground	Subwindow mode
Stipple	Font
Background	Tile
Tile stipple x origin	Tile stipple y origin
Clip x origin	Clip y origin
Clip mask	Fill style

To draw both foreground and background values of text, use the DRAW IMAGE STRING and DRAW IMAGE STRING 16 routines. For example, the sample program uses the DRAW IMAGE routine to write the text "Click Here to Exit," as follows:

```
int n, state = 0;
char *message[]= {
    "Click here to exit",
    "Click HERE to exit!"
    };
   .
   .
   .
 if (eventP->xexpose.window != window2) return;
 XDrawImageString(dpy, window2, gc, 75, 75, message[state],
     strlen(message[state]));
```

The effect is first to fill a rectangle with the background defined in the graphics context and then to paint the text with the foreground pixel. The upper left corner of the filled rectangle is at 75, (75 − *font ascent*). The width of the rectangle is equal to the width of the string. The height of the rectangle is equal to *font ascent* + *font descent*.

When drawing text in response to calls to DRAW IMAGE STRING and DRAW IMAGE STRING 16, the server ignores the function and fill style the client has defined in the graphics context. The value of the function member of the GC data structure is effectively the value specified by the constant **GXCopy**. The value of the fill style member is effectively the value specified by the constant **FillSolid**.

The server refers to the following members of the GC data structure when writing text with DRAW IMAGE STRING and DRAW IMAGE STRING 16:

Subwindow mode	Plane mask
Foreground	Background
Stipple	Font
Clip x origin	Clip y origin
Clip mask	

9 Handling Events

An event is a report of either a change in the state of a device (such as a mouse) or the execution of a routine called by a client. An event can be either unsolicited or solicited. Typically, unsolicited events are reports of keyboard or pointer activity. Solicited events are Xlib responses to calls by clients.

Xlib reports events asynchronously. When any event occurs, Xlib processes the event and sends it to clients that have specified an interest in that type of event.

This chapter describes the following concepts needed to manage events:

- Event processing—An overview of types of events
- Event type selection—A description of how clients can specify the types of events Xlib reports to them
- Event handling—A description of handling specific types of events

9.1 Event Processing

Apart from errors, which Section 9.13 describes, Xlib events issue from operations on either windows or pixmaps. Most events result from operations associated with windows. The smallest window that contains the pointer when a window event occurs is the **source window**.

Xlib searches the window hierarchy upward from the source window until one of the following applies:

- Xlib finds a window that one or more clients has identified as interested in the event. This window is the **event window**. After Xlib locates an event window, it sends information about the event to appropriate clients.
- Xlib finds a window whose do_not_propagate attribute has been set by a client. Setting this attribute specifies that Xlib should not notify ancestors of the window owned by the client about events occurring in the window and its children. For more information about the do_not_ propagate attribute, see Chapter 3.
- Xlib reaches the top of the window hierarchy without finding a window that a client has identified as interested in the event.

While there are many types of window events, events associated with pixmaps occur only when a client cannot compute a destination region because the source region is out of bounds (see Chapter 6 for a description of source and destination regions). When a client attempts an operation on an out of bounds pixmap region, Xlib puts the event on the event queue and checks a list to determine if a client is interested in the event. If a client is interested, Xlib sends information to the client using an event data structure.

Xlib can report 30 types of events related to keyboards, mice, windowing, and graphics operations. A flag identifies each type to facilitate referring to the event. Table 9–1 lists event types, grouped by category, and the flags that represent them.

Table 9–1 Event Types

Event Type	Flag Name
Keyboard Events	
Key press	KeyPress
Key release	KeyRelease
Pointer Motion Events	
Button press	ButtonPress
Button release	ButtonRelease
Motion notify	MotionNotify
Window Crossing Events	
Enter notify	EnterNotify
Leave notify	LeaveNotify
Input Focus Events	
Focus in	FocusIn
Focus out	FocusOut
Keymap State Event	
Keymap notify	KeymapNotify
Exposure Events	
Expose	Expose
Graphics expose	GraphicsExpose
No expose	NoExpose

(continued on next page)

Table 9–1 (Cont.) Event Types

Event Type	Flag Name
Window State Events	
Circulate notify	CirculateNotify
Configure notify	ConfigureNotify
Create notify	CreateNotify
Destroy notify	DestroyNotify
Gravity notify	GravityNotify
Map notify	MapNotify
Mapping notify	MappingNotify
Reparent notify	ReparentNotify
Unmap notify	UnmapNotify
Visibility notify	VisibilityNotify
Color Map State Events	
Color map notify	ColormapNotify
Client Communication Events	
Client message	ClientMessage
Property notify	PropertyNotify
Selection clear	SelectionClear
Selection notify	SelectionNotify
Selection request	SelectionRequest

Every event type has a corresponding data structure that Xlib uses to pass information to clients. See the sections that describe handling specific event types for a description of the relevant event-specific data structures.

Xlib includes the any event data structure, which clients can use to receive reports of any type of event. The following illustrates the any event data structure:

```
typedef struct {
        int type;
        unsigned long serial;
        Bool send_event;
        Display *display;
        Window window;
} XAnyEvent;
```

Table 9–2 describes members of the data structure.

Table 9–2 Any Event Data Structure Members

Member Name	Contents
type	Type of event being reported
serial	Number of the last request processed by the server
send_event	Value defined by the constant true if the event came from a SEND EVENT request
display	Display on which the event occurred
window	Window on which the event report was requested

To enable clients to manage multiple types of events easily, Xlib also includes an event data structure, which is composed of the union of individual event data structures.

The following illustrates the event data structure:

```
typedef union _XEvent {
        int type;
        XAnyEvent xany;
        XKeyEvent xkey;
        XButtonEvent xbutton;
        XMotionEvent xmotion;
        XCrossingEvent xcrossing;
        XFocusChangeEvent xfocus;
        XExposeEvent xexpose;
        XGraphicsExposeEvent xgraphicsexpose;
        XNoExposeEvent xnoexpose;
        XVisibilityEvent xvisibility;
        XCreateWindowEvent xcreatewindow;
        XDestroyWindowEvent xdestroywindow;
        XUnmapEvent xunmap;
        XMapEvent xmap;
        XMapRequestEvent xmaprequest;
        XReparentEvent xreparent;
        XConfigureEvent xconfigure;
        XGravityEvent xgravity;
        XResizeRequestEvent xresizerequest;
        XConfigureRequestEvent xconfigurerequest;
        XCirculateEvent xcirculate;
        XCirculateRequestEvent xcirculaterequest;
        XPropertyEvent xproperty;
        XSelectionClearEvent xselectionclear;
        XSelectionRequestEvent xselectionrequest;
        XSelectionEvent xselection;
        XColormapEvent xcolormap;
        XClientMessageEvent xclient;
        XMappingEvent xmapping;
        XErrorEvent xerror;
        XKeymapEvent xkeymap;
} XEvent;
```

The type member specifies the type of event being reported. For descriptions of the other members of the event data structure, see the section that describes the specific event type.

9.2 Selecting Event Types

Xlib sends information about an event only to clients that have specified an interest in that event type. Clients use one of the following methods to indicate interest in event types:

- By calling the SELECT INPUT routine. SELECT INPUT indicates to Xlib which events to report.
- By specifying event masks when creating a window.
- By specifying event masks when changing window attributes.
- By specifying the graphics exposure mask when creating the graphics context. For more information about specifying a graphics exposure mask, see Chapter 4.

Note that Xlib always reports client messages, mapping notifications, selection clearings, selection notifications, and selection requests.

See the description of the SELECT INPUT routine in the *VMS DECwindows Xlib Routines Reference Manual* for restrictions on event reporting to multiple clients.

9.2.1 Using the SELECT INPUT Routine

Use the SELECT INPUT routine to specify the types of events Xlib reports to a client. Select event types by passing to Xlib one or more of the masks listed in Table 9–3.

Table 9–3 Event Masks

Event Mask	Event Reported (Event Type)
ButtonMotionMask	At least one button on the pointing device is pressed while the pointer moves (MotionNotify).
Button1MotionMask	Button 1 of the pointing device is pressed while the pointer moves (MotionNotify).
Button2MotionMask	Button 2 of the pointing device is pressed while the pointer moves (MotionNotify).
Button3MotionMask	Button 3 of the pointing device is pressed while the pointer moves (MotionNotify).
Button4MotionMask	Button 4 of the pointing device is pressed while the pointer moves (MotionNotify).
Button5MotionMask	Button 5 of the pointing device is pressed while the pointer moves (MotionNotify).
ButtonPressMask	A button on the pointing device is pressed (ButtonPress).
ButtonReleaseMask	A button on the pointing device is released (ButtonRelease).
ColormapChangeMask	A client installs, changes, or removes a color map (ColormapNotify).
EnterWindowMask	The pointer enters a window (EnterNotify).
ExposureMask	A window becomes visible, a graphics region cannot be computed, a graphics request exposes a region, or all sources available and a no expose event generated (Expose, GraphicsExpose, NoExpose).
LeaveWindowMask	The pointer leaves a window (LeaveNotify).

(continued on next page)

Table 9–3 (Cont.) Event Masks

Event Mask	Event Reported (Event Type)
FocusChangeMask	The keyboard focus changes (FocusIn, FocusOut).
KeymapStateMask	The key map changes (KeymapNotify).
KeyPressMask	A key is pressed or released (KeyPress, KeyRelease).
OwnerGrabButtonMask	Not applicable.
PointerMotionMask	The pointer moves (MotionNotify).
PointerMotionHintMask	Xlib is free to report only one pointer-motion event (MotionNotify) until one of the following occurs: • Either the key or button state changes. • The pointer leaves the window. • The client calls QUERY POINTER or GET MOTION EVENTS.
PropertyChangeMask	A client changes a property (PropertyNotify).
StructureNotifyMask	One of the following operations occurs on a window: • Circulate (CirculateNotify) • Configure (ConfigureNotify) • Destroy (DestroyNotify) • Move (GravityNotify) • Map (MapNotify) • Reparent (ReparentNotify) • Unmap (UnmapNotify)
SubstructureNotifyMask	One of the following operations occurs on the child of a window: • Circulate (CirculateNotify) • Configure (ConfigureNotify) • Create (CreateNotify) • Destroy (DestroyNotify) • Move (GravityNotify) • Map (MapNotify) • Reparent (ReparentNotify) • Unmap (UnmapNotify)
VisibilityChangeMask	The visibility of a window changes (VisibilityNotify).

The following illustrates using the SELECT INPUT routine:

```
.
.
.
XSelectInput(dpy,win,StructureNotifyMask);
}
```

Clients specify the **StructureNotifyMask** mask to indicate an interest in one or more of the following window operations (see Table 9–3):

Circulating	Configuring
Destroying	Reparenting
Changing gravity	Mapping and unmapping

9.2.2 Specifying Event Types When Creating a Window

To specify event types when calling the CREATE WINDOW routine, use the method described in Section 3.2.2 for setting window attributes. Indicate the type of event Xlib reports to a client by doing the following:

1. Set the event_mask window attribute to one or more masks listed in Table 9–3.
2. Specify the event mask flag using the **value_mask** argument of the CREATE WINDOW routine.

Example 9–1 illustrates this method of selecting events. The program specifies that Xlib notify the client of exposure events.

Example 9–1 Selecting Event Types Using the CREATE WINDOW Routine

```
Window window;
    .
    .
    .
static void doCreateWindows( )
{
    int windowW = 400;
    int windowH = 300;
    int windowX = (WidthOfScreen(screen)-window1W)>>1;
    int windowY = (HeightOfScreen(screen)-window1H)>>1;
    XSetWindowAttributes xswa;

    /* Create the window1 window */
❶    xswa.event_mask = ExposureMask;

❷    window = XCreateWindow(dpy, RootWindowOfScreen(screen),
         windowX, windowY, windowW, windowH, 0,
         DefaultDepthOfScreen(screen), InputOutput,
         DefaultVisualOfScreen(screen), CWEventMask, &xswa);
}
```

❶ Set the event mask of the set window attributes data structure to indicate interest in exposure events.

❷ The window attribute is referred to by the constant **CWEventMask**, which specifies the attribute.

9.2.3 Specifying Event Types When Changing Window Attributes

To specify one or more event types when changing window attributes, use the method described in Section 3.6 for changing window attributes. Indicate an interest in event types by doing the following:

1. Set the event_mask window attribute to one or more masks listed in Table 9–3.
2. Specify the event mask flag using the **value_mask** argument of the CHANGE WINDOW ATTRIBUTES routine.

The following illustrates this method:

```
.
.
.
xswa.event_mask = StructureNotify;
XChangeWindowAttributes(dpy, win, CWEventMask, &xswa);
```

9.3 Pointer Events

Xlib reports pointer events to interested clients when the button on the pointing device is pressed or released, or when the pointer moves.

This section describes how to handle the following pointer events:

- Pressing a button on the pointing device
- Releasing a button on the pointing device
- Moving the pointing device

The section also describes the button event and motion event data structures.

9.3.1 Handling Button Presses and Releases

To receive event notification of button presses and releases, pass the window identifier and either the **ButtonPressMask** mask or the **ButtonReleaseMask** mask when using the selection method described in Section 9.2.

When a button is pressed, Xlib searches for ancestors of the event window from the root window down to determine whether or not a client has specified a **passive grab**, an exclusive interest in the button. If Xlib finds no passive grab, it starts an **active grab**, reserving the button for the sole use of the client receiving notification of the event. Xlib also sets the time of the last pointer grab to the current Xlib time. The effect is the same as calling the GRAB BUTTON routine with argument values listed in Table 9–4.

Table 9–4 Values Used for Grabbing Buttons

Argument	Value
window_id	Event window.
event_mask	Client pointer motion mask.
pointer_mode	Value specified by the constant GrabModeAsync.
keyboard_mode	Value specified by the constant GrabModeAsync.

(continued on next page)

Table 9–4 (Cont.) Values Used for Grabbing Buttons

Argument	Value
owner_events	True, if the owner has selected OwnerGrabButtonMask. Otherwise, false.
confine_to	None.
cursor	None.

Xlib terminates the grab automatically when the button is released. Clients can modify the active grab by calling the UNGRAB POINTER and CHANGE ACTIVE POINTER GRAB routines.

Xlib uses the button event data structure to report button presses and releases. The following illustrates the data structure:

```
typedef struct {
        int type;
        unsigned long serial;
        Bool send_event;
        Display *display;
        Window window;
        Window root;
        Window subwindow;
        Time time;
        int x, y;
        int x_root, y_root;
        unsigned int state;
        unsigned int button;
        Bool same_screen;
} XButtonEvent;
typedef XButtonEvent XButtonPressedEvent;
typedef XButtonEvent XButtonReleasedEvent;
```

Table 9–5 describes members of the button event data structure. Note that Xlib defines the button pressed event and button released event data structures as a type button event.

Table 9–5 Button Event Data Structure Members

Member Name	Contents
type	Type of event reported. The event type can be either ButtonPress or ButtonRelease.
serial	Number of the last request processed by the server.
send_event	Value defined by the constant true if the event came from a SEND EVENT request.
display	Address of the display on which the event occurred.
window	Event window.
root	Root window in which the event occurred.
subwindow	Source window in which the event occurred.

(continued on next page)

Table 9–5 (Cont.) Button Event Data Structure Members

Member Name	Contents	
time	Time in milliseconds at which the event occurred.	
x	The x value of the pointer coordinates in the source window.	
y	The y value of the pointer coordinates in the source window.	
x_root	The x value of the pointer coordinates relative to the root window.	
y_root	The y value of the pointer coordinates relative to the root window.	
state	State of the button just prior to the event. Xlib can set this member to the bitwise OR of one or more of the following masks:	
	Button1Mask	Button2Mask
	Button3Mask	Button4Mask
	Button5Mask	Mod1Mask
	Mod2Mask	Mod3Mask
	Mod4Mask	Mod5Mask
button	Buttons that changed state. Xlib can set this member to one of the following values:	
	Button1	Button2
	Button3	Button4
	Button5	
screen	Indicates whether or not the event window is on the same screen as the root window.	

Example 9–2 illustrates the button press event handling routine of the sample program described in Chapter 1. The program calls shutdown routines when the user clicks the mouse button in *window2*.

Xlib removes the next event from the event queue and copies it into an event data structure. The program executes one of two routines, depending on the flag returned in the event data structure type field. Xlib indicates an exposure event by setting the Expose flag in the type field; it indicates a button press event by setting the ButtonPress flag.

When creating *window1* and *window2*, the client indicated an interest in exposures and button presses by setting the event mask field of the set window attributes data structure, as follows:

```
XSetWindowAttributes xswa;
      .
      .
      .
xswa.event_mask = ExposureMask | ButtonPressMask;
```

For more information about selecting event types, see Section 9.2.

Example 9–2 Handling Button Presses

```
/***************** doHandleEvents **********************/
static void doHandleEvents( )
{
    XEvent event;

    for ( ; ; ) {
        XNextEvent(dpy, &event);
        switch (event.type) {
            case Expose:                doExpose(&event); break;
            case ButtonPress:           doButtonPress(&event); break;
        }
    }
}
   .
   .
   .
static void doButtonPress(eventP)
XEvent *eventP;
{
        if (eventP->xexpose.window != window2) {
        state =1;
        XDrawImageString(dpy, child, gc, 75, 75, message[state],
            strlen(message[state]));
        return;
    }

    /* Unmap and destroy windows */

    XUnmapWindow(dpy, window1);
    XDestroyWindow(dpy, window1);

    XCloseDisplay(dpy);

    sys$exit (1);
}
```

The event data structure includes other data structures Xlib uses to report information about various kinds of events. The client-defined *doButtonPress* routine checks the window field of one of these data structures (the expose event data structure) to determine whether or not the server has mapped *window2*.

If the server has mapped *window2*, the client calls a series of shutdown routines when the user presses the mouse button.

9.3.2 Handling Pointer Motion

To only receive pointer motion events when a specified button is pressed, pass the window identifier and one of the following masks when using the selection method described in Section 9.2:

ButtonMotionMask	Button1MotionMask
Button2MotionMask	Button3MotionMask
Button4MotionMask	Button5MotionMask

Xlib reports pointer motion events to interested clients whenever the pointer moves and the movement begins and ends in the window. Spatial and temporal resolution of the events is not guaranteed, but clients are assured they will receive at least one event when the pointer moves and then rests. The following illustrates the data structure Xlib uses to report these events:

```
typedef struct {
        int type;
        unsigned long serial;
        Bool send_event;
        Display *display;
        Window window;
        Window root;
        Window subwindow;
        Time time;
        int x, y;
        int x_root, y_root;
        unsigned int state;
        char is_hint;
        Bool same_screen;
} XMotionEvent;
typedef XMotionEvent XPointerMovedEvent;
```

Table 9–6 describes members of the motion event data structure. Note that Xlib defines the pointer moved event data structure as type motion event.

Table 9–6 Motion Event Data Structure Members

Member Name	Contents
type	Type of event reported. The member can have only the value specified by the constant MotionNotify.
serial	Number of the last request processed by the server.
send_event	Value defined by the constant true if the event came from a SEND EVENT request.
display	Address of the display on which the event occurred.
window	Event window.
root	Root window in which the event occurred.
subwindow	Source window in which the event occurred.
time	Time in milliseconds at which the event occurred.
x	The x value of the pointer coordinates in the source window.
y	The y value of the pointer coordinates in the source window.
x_root	The x value of the pointer coordinates relative to the root window.
y_root	The y value of the pointer coordinates relative to the root window.

(continued on next page)

Table 9–6 (Cont.) Motion Event Data Structure Members

Member Name	Contents	
state	State of the button just prior to the event. Xlib can set this member to the bitwise OR of the following masks:	
	Button1Mask	Button2Mask
	Button3Mask	Button4Mask
	Button5Mask	Mod1Mask
	Mod2Mask	Mod3Mask
	Mod4Mask	Mod5Mask
is_hint	Indicates that motion hints are active. No other events reported until pointer moves out of window.	
same_screen	Indicates whether or not the event window is on the same screen as the root window.	

Example 9–3 illustrates pointer motion event handling.

Example 9–3 Handling Pointer Motion

```
/****************** doHandleEvents ***********************/
static void doHandleEvents( )
{
    XEvent event;

    for ( ; ; ) {
        XNextEvent(dpy, &event);
        switch (event.type) {
            case Expose:               doExpose(&event); break;
            case MotionNotify:         doMotionNotify(&event); break;
            case ButtonPress:          sys$exit(1);
        }
    }
}
    .
    .
    .
static void doMotionNotify(eventP)
XEvent *eventP;
{
    int x = eventP->xmotion.x;
    int y = eventP->xmotion.y;
    int width = 5;
    int length = 5;

    XFillRectangle(dpy, win, gc, x, y, width, length);
}
```

Each time the pointer moves, the program draws a small filled rectangle at the resulting *x* and *y* coordinates.

To receive pointer motion events, the client specifies the **MotionNotify** flag when removing events from the queue. The client indicated an interest in pointer motion events when creating window *win*, as follows:

```
    xswa.event_mask = ExposureMask | ButtonPressMask |
        PointerMotionMask;

    win = XCreateWindow(dpy, RootWindowOfScreen(screen),
        winX, winY, winW, winH, 0,
        DefaultDepthOfScreen(screen), InputOutput,
        DefaultVisualOfScreen(screen), CWEventMask, &xswa);
```

The server reports pointer movement. Xlib records the resulting position of the pointer in a motion data structure, one of the event data structures that constitute the event data structure. The client-defined *doMotionNotify* routine determines the origin of the filled rectangle it draws by referring to the motion event data structure *x* and *y* members.

9.4 Key Events

Xlib reports key press and key release events to interested clients. To receive event notification of key presses and releases, pass the window identifier and either the **KeyPressMask** mask or the **KeyReleaseMask** mask when using the selection method described in Section 9.2.

Xlib uses a key event data structure to report key presses and releases to interested clients whenever any key changes state, even when the key is mapped to modifier bits.

The following illustrates the data structure:

```
typedef struct {
        int type;
        unsigned long serial;
        Bool send_event;
        Display *display;
        Window window;
        Window root;
        Window subwindow;
        Time time;
        int x, y;
        int x_root, y_root;
        unsigned int state;
        unsigned int keycode;
        Bool same_screen;
} XKeyEvent;
typedef XKeyEvent XKeyPressedEvent;
typedef XKeyEvent XKeyReleasedEvent;
```

Table 9–7 describes members of the key event data structure. Note that Xlib defines the key pressed event and key released event data structures as type key event.

Table 9–7 Key Event Data Structure Members

Member Name	Contents
type	Value defined by either the KeyPress or the KeyRelease constant.
serial	Number of the last request processed by the server.
send_event	Value defined by the constant true if the event came from a SEND EVENT request.
display	Address of the display on which the event occurred.
window	Event window.
root	Root window on which the event occurred.
subwindow	Source window of the event.
time	Time in milliseconds at which the key event occurred.
x	The x value of the pointer coordinates in the source window.
y	The y value of the pointer coordinates in the source window.
x_root	The x value of the pointer coordinates relative to the root window.
y_root	The y value of the pointer coordinates relative to the root window.
state	State of the key just prior to the key event. Xlib can set this member to the bitwise OR of the the following states: ShiftMask, LockMask, ControlMask, Mod1Mask, Mod2Mask, Mod3Mask, Mod4Mask, Mod5Mask
keycode	An arbitrary but unique representation of the key that generated the event.
same_screen	Indicates whether the event window is on the same screen as the root window.

9.5 Window Entries and Exits

Xlib reports window entries and exits to interested clients when one of the following occurs:

- The pointer moves into or out of a window due to either pointer movement or to a change in window hierarchy. This is normal window entry and exit.
- A client calls WARP POINTER, which moves the pointer to any specified point on the screen.
- A client calls CHANGE ACTIVE POINTER GRAB, GRAB KEYBOARD, GRAB POINTER, or UNGRAB POINTER. This is **pseudomotion**, which simulates window entry or exit without actual pointer movement.

To receive event notification of window entries and exits, pass the window identifier and either the **EnterWindowMask** mask or the **LeaveWindowMask** mask when using the selection method described in Section 9.2.

Xlib uses the crossing event data structure to report window entries and exits. The following illustrates the data structure:

```
typedef struct {
        int type;
        unsigned long serial;
        Bool send_event;
        Display *display;
        Window window;
        Window root;
        Window subwindow;
        Time time;
        int x, y;
        int x_root, y_root;
        int mode;
        int detail;
        Bool same_screen;
        Bool focus;
        unsigned int state;
} XCrossingEvent;
typedef XCrossingEvent XEnterWindowEvent;
typedef XCrossingEvent XLeaveWindowEvent;
```

Table 9–8 describes members of the crossing event data structure. Note that Xlib defines the enter window event and leave window event data structures as type crossing event.

Table 9–8 Crossing Event Data Structure Members

Member Name	Contents
type	Value defined by either the EnterNotify or the LeaveNotify constant.
serial	Number of the last request processed by the server.
send_event	The value defined by the constant true if the event came from a SEND EVENT request.
display	Address of the display on which the event occurred.
window	Event window.
root	Root window on which the event occurred.
subwindow	Source window in which the event occurred.
time	Time in milliseconds at which the key event occurred.
x	The x value of the pointer coordinates in the source window.
y	The y value of the pointer coordinates in the source window.
x_root	The x value of the pointer coordinates relative to the root window.

(continued on next page)

Table 9–8 (Cont.) Crossing Event Data Structure Members

Member Name	Contents
y_root	The y value of the pointer coordinates relative to the root window.
mode	Indicates whether the event is normal or pseudomotion. Xlib can set this member to the value specified by the constants NotifyNormal, NotifyGrab, and NotifyUngrab. See Section 9.5.1 and Section 9.5.2 for descriptions of normal and pseudomotion events.
detail	Indicates which windows Xlib notifies of the window entry or exit event. Xlib can specify one of the following constants: NotifyAncestor, NotifyVirtual, NotifyInferior, NotifyNonlinear, NotifyNonlinearVirtual
same_screen	Indicates whether or not the event window is on the same screen as the root window.
focus	Specifies whether the event window or an inferior is the focus window. If true, the event window is the focus window. If false, an inferior is the focus window.
state	State of buttons and keys just prior to the event. Xlib can return values specified by the following constants: Button1Mask, Button2Mask, Button3Mask, Button4Mask, Button5Mask, Mod1Mask, Mod2Mask, Mod3Mask, Mod4Mask, Mod5Mask, ShiftMask, ControlMask, LockMask

9.5.1 Normal Window Entries and Exits

A normal window entry or exit event occurs when the pointer moves from one window to another due to either a change in window hierarchy or the movement of the pointer. In either case, Xlib sets the mode member of the crossing event data structure to the constant **NotifyNormal**.

If the pointer leaves or enters a window as a result of one of the following changes in window hierarchy, Xlib reports the event after reporting the hierarchy event:

Mapping	Unmapping
Configuring	Circulating
Changing gravity	

Xlib can report a window entry or exit event caused by changes in focus, visibility, and exposure either before or after reporting these events.

Table 9–9 describes the events Xlib reports when the pointer moves from window A to window B as a result of a normal window entry or exit.

Table 9–9 Normal Window Entry and Exit Event Reporting

Relationship of Windows	Events Reported
Window A is inferior to window B	A leave notify event on window A with the detail member of the crossing event data structure set to the constant NotifyAncestor
	A leave notify event on each window between window A and window B exclusive, with the detail member of each crossing event data structure set to the constant NotifyVirtual
	An enter notify event on window B with the detail member of the crossing event data structure set to the constant NotifyInferior
Window B is inferior of window A	A leave notify event on window A with the detail member of the crossing event data structure set to the constant NotifyInferior
	An enter notify event on each window between window A and window B exclusive with the detail member of each crossing event data structure set to the constant NotifyVirtual
	An enter notify event on window B with the detail member of the crossing event data structure set to the constant NotifyAncestor
Window C is the least common ancestor of A and B	A leave notify event on window A with the detail member of the crossing event data structure set to the constant NotifyNonlinear
	A leave notify event on each window between window A and window C exclusive with the detail member of the crossing event data structure set to the constant NotifyNonlinearVirtual
	An enter notify event on each window between window C and window B exclusive with the detail member of each crossing event data structure set to the constant NotifyNonlinearVirtual
	An enter notify event on window B with the detail member of the crossing event data structure set to the constant NotifyNonlinear
Window A and window B are on different screens	A leave notify event on window A with the detail member of the crossing event data structure set to the constant NotifyNonlinear
	If window A is not a root window, a leave notify event on each window above window A up to and including its root, with the detail member of each crossing event data structure set to the constant NotifyNonlinearVirtual
	If window B is not a root, an entry notify event on each window from window B's root down to but not including window B, with the detail member of the crossing event data structure set to the constant NotifyNonlinearVirtual
	An enter notify event on window B with the detail member of the crossing event data structure set to the constant NotifyNonlinear

Example 9–4 illustrates window entry and exit event handling. The program changes the color of a window when the pointer enters or leaves the window.

Figure 9–1 shows the resulting output.

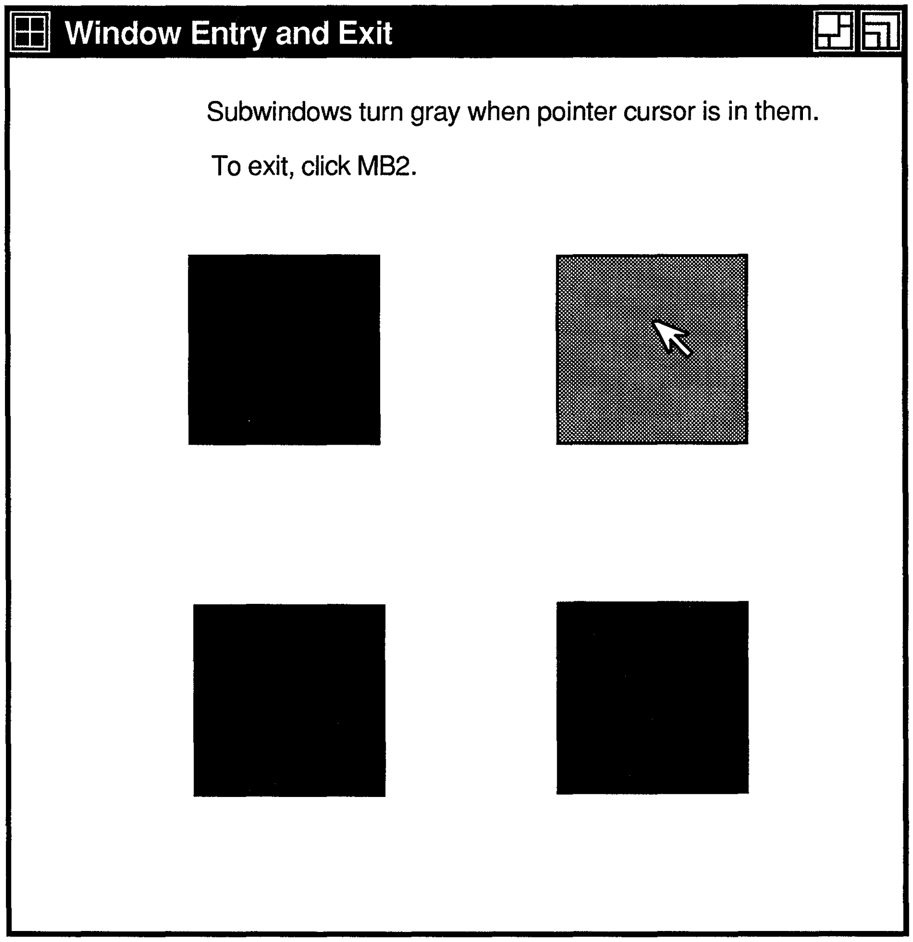
Example 9–4 Handling Window Entries and Exits

```
    /* Create windows win, subwin1,              *
     * subwin2, subwin3, and                     *
     * subwin4  on                               *
     *  display dpy, defined as follows:         *
     *     #define windowWidth      600          *
     *     #define windowHeight     600          *
     *     #define subwindowWidth  120           *
     *     #define subwindowHeight 120           *
     *     win position: x = 100,y = 100         *
     *     subwin1 position: x = 120,y = 120     *
     *     subwin2 position: x = 360,y = 120     *
     *     subwin3 position: x = 120,y = 360     *
     *     subwin1 position: x = 360,y = 360     */
   .
   .
   .
/****************** doHandleEvents **********************/
static void doHandleEvents( )
{
    XEvent event;

    for ( ; ; ) {
        XNextEvent(dpy, &event);
        switch (event.type) {
            case Expose:                doExpose(&event); break;
            case ButtonPress:           sys$exit(1);
            case EnterNotify:           doEnterNotify(&event); break;
            case LeaveNotify:           doLeaveNotify(&event); break;
        }
    }
}
   .
   .
   .
/***** Change window color when pointer enters window *******/
static void doEnterNotify(eventP)
XEvent *eventP;
{
❶   Window window = eventP->xcrossing.window;
    XSetWindowBackground(dpy, window, doDefineColor(4));
❷   XClearArea(dpy, window, 0, 0, subwindowWidth, subwindowHeight, 0);
    return;
}

/***** Change window color when pointer leaves window *******/
static void doLeaveNotify(eventP)
XEvent *eventP;
{
    Window window = eventP->xcrossing.window;
    XSetWindowBackground(dpy, window, doDefineColor(2));
    XClearArea(dpy, window, 0, 0, subwindowWidth, subwindowHeight, 0);
    return;
}
```

❶ Xlib gives the identifier of the window that the pointer cursor has entered in the crossing event data structure window field. The program uses the identifier to define the window background and clear the window.

❷ The CLEAR AREA routine clears the window and repaints it with the newly defined window background.

Figure 9–1 Window Entries and Exits



Subwindows turn gray when pointer cursor is in them.

To exit, click MB2.

ZK–0153A–GE

9.5.2 Pseudomotion Window Entries and Exits

Pseudomotion window entry and exit events occur when the pointer cursor moves from one window to another due to activating or deactivating a pointer grab.

Xlib reports a pseudomotion window entry if a client grabs the pointer, causing the pointer cursor to change from one window to another even though the pointer cursor has not moved. For example, if the pointer cursor is in window A and a client maps window B over window A, the pointer cursor changes from being in window A to being in window B. If possible, the pointer cursor remains in the same position on the screen. When the placement of the two windows prevents the pointer cursor from maintaining the same position, the pointer cursor moves to the location closest to its original position.

Clients can grab pointers actively by calling the GRAB POINTER routine or passively by calling the GRAB BUTTON routine. Whether the grab is active or passive, Xlib sets the following members of the crossing event data structure to the indicated constants after the pointer cursor moves from one window to another:

- Type member—EnterNotify
- Mode member—NotifyGrab

When a client passively grabs the pointer by calling the GRAB BUTTON routine, Xlib reports a button press event after reporting the pointer grab.

Xlib reports a pseudomotion window exit when a client deactivates a pointer grab, causing the pointer cursor to change from one window to another even though the pointer cursor has not moved.

Clients can deactivate pointer grabs either actively by calling the UNGRAB POINTER routine or passively by calling the UNGRAB BUTTON routine. Whether deactivating the grab is active or passive, Xlib sets the following members of the crossing event data structure to the indicated constants after the pointer cursor moves from one window to another:

- Type member—LeaveNotify
- Mode member—NotifyUngrab

When a client passively deactivates a pointer grab by calling the UNGRAB BUTTON routine, Xlib reports a button release event before reporting that the pointer has been released.

9.6 Input Focus Events

Input focus defines the window to which Xlib sends keyboard input. The keyboard is always attached to some window. Typically, keyboard input goes to either the root window or to a window at the top of the stack called the **focus window**. The focus window and the position of the pointer determine the window that receives keyboard input.

When the keyboard input focus changes from one window to another, Xlib reports a focus out event and a focus in event. The window that loses the input focus receives the focus out event. The window that gains the focus receives a focus in event. Additionally, Xlib notifies other windows in the hierarchy of focus in and focus out events.

To receive notification of input focus events, pass the window identifier and the **FocusChangeMask** mask when using the selection method described in Section 9.2.

Xlib uses the focus change event data structure to report keyboard input focus events. The following illustrates the data structure:

```
typedef struct {
        int type;
        unsigned long serial;
        Bool send_event;
        Display *display;
        Window window;
        int mode;
        int detail;
} XFocusChangeEvent;
typedef XFocusChangeEvent XFocusInEvent;
typedef XFocusChangeEvent XFocusOutEvent;
```

Table 9–10 describes members of the focus change event data structure. Note that Xlib defines the focus in event and focus out event data structures as type focus change event.

Table 9–10 Focus Change Event Data Structure Members

Member Name	Contents
type	Value defined by either the FocusIn or the FocusOut constant.
serial	Number of the last request processed by the server.
send_event	Value defined by the constant true if the event came from a SEND EVENT request.
display	Address of the display on which the event occurred.
window	Event window.
mode	Specifies whether the event is the result of normal keyboard input, keyboard input after a client has grabbed the keyboard, keyboard input at the time the client activates a keyboard grab, or keyboard input at the time the client deactivates a keyboard grab. Xlib can set this field to one of the following constants: NotifyNormal NotifyWhileGrabbed NotifyGrab NotifyUngrab See Section 9.6.1 and Section 9.6.2 for descriptions of processing input focus events in each of these conditions.
detail	Indicates which windows and pointers Xlib notifies of the input focus change. Xlib can set this field to one of the following constants: NotifyAncestor NotifyVirtual NotifyInferior NotifyNonLinear NotifyNonLinearVirtual NotifyPointer NotifyPointerRoot NotifyDetailNone

9.6.1 Normal Keyboard Input Focus

A normal keyboard input focus event occurs when keyboard input focus changes, and the keyboard has not been or is not being grabbed. When a normal keyboard input focus event occurs, Xlib sets the MODE member of the focus change event data structure to the constant **NotifyNormal**.

Table 9–11 lists focus change events reported when window A and window B are on the same screen, the focus changes from window A to window B, and the pointer cursor is in window P.

Table 9–11 Effect of Focus Changes: Windows on Same Screen

Window A Inferior to Window B

Window	Event Reported	Value of DETAIL
Window A	Focus out event	NotifyAncestor
Window B	Focus in event	NotifyInferior
Window P	Focus in event on each window between window B and window P including P if window P is inferior of window B, but window P is not window A or an inferior of A	NotifyInferior
Other windows	Focus out event on each window between window A and window B exclusive	NotifyVirtual

Window B Inferior to Window A

Window	Event Reported	Value of DETAIL
Window A	Focus out event	NotifyInferior
Window B	Focus in event	NotifyAncestor
Window P	Focus out event on each window between window P and window A if window P is an inferior of window A, but window P is not window A or an inferior or ancestor of B	NotifyPointer
Other windows	Focus in event on each window between window A and window B exclusive	NotifyVirtual

Table 9–12 lists focus change events reported when the pointer cursor moves from window A to window B and window C is their least common ancestor. The pointer cursor is in window P.

Table 9–12 Focus Changes Caused by Pointer Movement

Pointer Moves From Window A to Window B

Window	Event Reported	Value of DETAIL
Window A	Focus out event	NotifyNonLinear
Window B	Focus in event	NotifyNonLinear
Window P	If window P is an inferior of window A, but window P is not window A or an inferior or ancestor of B, a focus out event on each window from window P up to but not including window A	NotifyPointer

(continued on next page)

Table 9–12 (Cont.) Focus Changes Caused by Pointer Movement

Pointer Moves From Window A to Window B		
Window	**Event Reported**	**Value of DETAIL**
	If window P is an inferior of window B, a focus in event on each window below window B down to and including window P	NotifyPointer
Other windows	Focus out event on each window between window A and window C exclusive	NotifyNonlinearVirtual
	Focus in event on each window between window C and window B exclusive	NotifyNonlinearVirtual

Table 9–13 lists focus change events reported when window A and window B are on different screens and the focus changes from window A to window B. The pointer cursor is in window P.

Table 9–13 Effect of Focus Changes: Windows on Different Screens

Focus Changes from Window A to Window B		
Window	**Event Reported**	**Value of DETAIL**
Window A	Focus out event	NotifyNonlinear
Window B	Focus in event	NotifyNonlinear
Window P	If window P is an inferior of window A, a focus out event on each window from window P up to but not including window A	NotifyPointer
	If window P is an inferior of window B, a focus in event on each window below window B down to and including window P	NotifyPointer
Other windows	If window A is not a root window, a focus out event on each window above window A up to and including its root	NotifyNonlinearVirtual
	If window B is not a root window, a focus in event on each window from the root window of B down to but not including B	NotifyNonlinearVirtual

Table 9–14 lists focus change events reported when the focus changes between window A and the pointer window, or when the focus is set to none (no focus).

Table 9–14 Pointer Window and No Focus Changes

Focus Changes from Window A to Pointer Window or to No Focus

Window	Event Reported	Value of DETAIL
Window A	Focus out event	NotifyNonlinear
All root windows	Focus in event	NotifyPointerRoot or NotifyDetailNone
Window P	If window P is an inferior of window A, a focus out event on each window from window P up to but not including window A	NotifyPointer
Other windows	If window A is not a root window, a focus out event on each window above window A up to and including its root	NotifyNonlinearVirtual
	If the new focus is the window under the pointer, a focus in event on each window from the root of window P down to and including window P	NotifyPointerRoot

Focus Changes from Pointer Window or No Focus to Window A

Window	Event Reported	Value of DETAIL
Window A	Focus in event	NotifyNonlinear
All root windows	Focus out event	NotifyPointerRoot or NotifyDetailNone
Window P	If window P is an inferior of window A, a focus in event on each window below window A down to and including P	NotifyPointer
	Focus out event on each window from window P up to and including the root of P	NotifyPointerRoot
Other windows	Focus out event on each window from window P up to and including the root of P	NotifyPointerRoot
	If window A is not a root window, a focus in event on each window from the root of window A down to but not including A	NotifyNonlinearVirtual

Focus Changes from Pointer Window to No Focus or from No Focus to Pointer Window

Window	Event Reported	Value of DETAIL
All root windows	Focus out event	NotifyPointerRoot or NotifyDetailNone
Old focus window	If the old focus was the window under the pointer, a focus out event on each window from window P up to and including the root of P	NotifyPointerRoot
New focus window	If the new focus is the window under the pointer, a focus in event on each window from the root of P down to and including P	NotifyPointerRoot

9.6.2 Keyboard Input Focus Changes Caused by Grabs

When a keyboard focus event occurs because a client activates a grab, Xlib sets the move member of the focus change event data structure to the constant **NotifyGrab**.

When a keyboard focus event occurs because a client deactivates a grab, Xlib sets the move member of the focus change event data structure to the constant **NotifyUngrab**.

9.7 Key Map State Events

Xlib reports changes in the state of the key map immediately after every enter notify and focus in event.

To receive notification of key map state events, pass the window identifier and the **KeymapStateMask** mask when using the selection method described in Section 9.2.

Xlib uses the keymap event data structure to report changes in the key map state. The following illustrates the data structure:

```
typedef struct {
        int type;
        unsigned long serial;
        Bool send_event;
        Display *display;
        Window window;
        char key_vector[32];
} XKeymapEvent;
```

Table 9–15 describes members of the keymap event data structure.

Table 9–15 Keymap Event Data Structure Members

Member Name	Contents
type	Value defined by the KeymapNotify constant.
serial	Number of the last request processed by the server.
send_event	Value defined by the constant true if the event came from a SEND EVENT request.
display	Address of the display on which the event occurred.
window	Event window.
key_vector	Bit vector of the keyboard. Each one bit indicates that the corresponding key is currently pressed. Byte N contains the bits for keys 8N to 8N+7 with the least significant bit representing key 8N.

9.8 Exposure Events

Xlib reports an exposure event when one of the following conditions occurs:

- A formerly obscured window or window region becomes visible.
- A destination region cannot be computed.

- A graphics request exposes one or more regions.

This section describes how to handle window exposures and graphics exposures.

9.8.1 Handling Window Exposures

A window exposure occurs when a formerly obscured window becomes visible again. Because Xlib does not guarantee to preserve the contents of regions when windows are obscured or reconfigured, clients are responsible for restoring the contents of the exposed window.

To receive notification of window exposure events, pass the window identifier and the **ExposureMask** mask when using the selection method described in Section 9.2. Xlib notifies clients of window exposures using the expose event data structure.

The following illustrates the data structure.:

```
typedef struct {
        int type;
        unsigned long serial;
        Bool send_event;
        Display *display;
        Window window;
        int x, y;
        int width, height;
        int count;
} XExposeEvent;
```

Table 9–16 describes members of the expose event data structure.

Table 9–16 Expose Event Data Structure Members

Member Name	Contents
type	Value defined by the Expose constant.
serial	Number of the last request processed by the server.
send_event	Value defined by the constant true if the event came from a SEND EVENT request.
display	Address of the display on which the event occurred.
window	Event window.
x	The x value of the coordinates that define the upper left corner of the region that is exposed. The coordinates are relative to the origin of the drawable.
y	The y value of the coordinates that define the upper left corner of the region that is exposed. The coordinates are relative to the origin of the drawable.

(continued on next page)

Table 9–16 (Cont.) Expose Event Data Structure Members

Member Name	Contents
width	Width of the exposed region.
height	Height of the exposed region.
count	Number of exposure events that are to follow. If Xlib sets the count to zero, no more exposure events follow for this window. Clients that do not want to optimize redisplay by distinguishing between subareas of its window can ignore all exposure events with nonzero counts and perform full redisplays on events with zero counts.

The following fragment from the sample program in Chapter 1 illustrates window exposure event handling:

```
/****************** doHandleEvents ***********************/
static void doHandleEvents( )
{
    XEvent event;

    for ( ; ; ) {
        XNextEvent(dpy, &event);
        switch (event.type) {
            case Expose:                doExpose(&event); break;
    .
    .
    .
static void doExpose(eventP)
XEvent *eventP;
{
    char message[] = {"Click here to exit"};

    if (eventP->xexpose.window != window2) return;

    XDrawImageString(dpy, window2, gc, 75, 75, message,
         strlen(message));
}
}
```

The program checks exposure events to verify that the server has mapped the second window. After the window is mapped, the program writes text into it.

The client-defined *doExpose* routine checks the window and count members of the expose event data structure to determine whether or not the server has completed mapping *window2*. If the window is mapped, the program writes the message "Click here to exit" in it.

9.8.2 Handling Graphics Exposures

Xlib reports graphics exposures when one of the following conditions occurs:

- A destination region could not be computed due to an obscured or out of bounds source region. For information about destination and source regions, see Chapter 6.

- A graphics request exposes one or more regions. If the request exposes more than one region, Xlib reports them continuously.

Instead of using the SELECT INPUT routine to indicate an interest in graphics exposure events, assign a value of true to the graphics_exposures member of the GC values data structure. Clients can set the value to true at the time they create a graphics context. If a graphics context exists, use the SET GRAPHICS EXPOSURES routine to set the value of the field. For information about creating a graphics context and using the SET GRAPHICS EXPOSURES routine, see Chapter 4.

Xlib uses the graphics expose event data structure to report graphics exposures. The following illustrates the data structure:

```
typedef struct {
        int type;
        unsigned long serial;
        Bool send_event;
        Display *display;
        Drawable drawable;
        int x, y;
        int width, height;
        int count;
        int major_code;
        int minor_code;
} XGraphicsExposeEvent;
```

Table 9–17 describes members of the graphics expose event data structure.

Table 9–17 Graphics Expose Event Data Structure Members

Member Name	Contents
type	Value defined by the GraphicsExpose constant.
serial	Number of the last request processed by the server.
send_event	Value defined by the constant true if the event came from a SEND EVENT request.
display	Address of the display on which the event occurred.
drawable	Drawable reporting the event.
x	The x value of the coordinates that define the upper left corner of the exposed region. The coordinates are relative to the origin of the drawable.
y	The y value of the coordinates that define the upper left corner of the exposed region. The coordinates are relative to the origin of the drawable.
width	Width of the exposed region.
height	Height of the exposed region.

(continued on next page)

Table 9–17 (Cont.) Graphics Expose Event Data Structure Members

Member Name	Contents
count	Number of exposure events that are to follow. If Xlib sets the count to zero, no more exposure events follow for this window.
major_code	Indicates whether the graphics request was a copy area or a copy plane.
minor_code	The value zero. Reserved for use by extensions.

Xlib uses the no expose event data structure to report when a graphics request that might have produced an exposure did not. The following illustrates the data structure:

```
typedef struct {
        int type;
        unsigned long serial;
        Bool send_event;
        Display *display;
        Drawable drawable;
        int major_code;
        int minor_code;
} XNoExposeEvent;
```

Table 9–18 describes members of the no expose event data structure.

Table 9–18 No Expose Event Data Structure Members

Member Name	Contents
type	Value defined by the NoExpose constant.
serial	Number of the last request processed by the server.
send_event	Value defined by the constant true if the event came from a SEND EVENT request.
display	Address of the display on which the event occurred.
drawable	Window or pixmap reporting the event.
major_opcode	Indicates whether the graphics request was a copy area or a copy plane.
minor_opcode	The value zero. Reserved for use by extensions.

Example 9–5 illustrates handling graphics exposure events. The program checks for graphics exposures and no exposures to scroll up a window.

Figure 9–2 shows the resulting output of the program.

Example 9–5 Handling Graphics Exposures

```
#define scrollPixels    1
#define windowWidth     600
#define windowHeight    600

Display *dpy;
Window win;
GC gc;
Screen *screen;
int n;
int ButtonIsDown;
int vY = 0;
   .
   .
   .

/****************** doHandleEvents **********************/
static void doHandleEvents( )
{
    XEvent event;

    for ( ; ; ) {
        XNextEvent(dpy, &event);
        switch (event.type) {
            case Expose:                doExpose(&event); break;
            case ButtonPress:           doButtonPress(&event); break;
            case GraphicsExpose:        doGraphicsExpose(&event); break;
            case ButtonRelease:         doButtonRelease(&event); break;
            case NoExpose:              doNoExpose(&event); break;
        }
    }
}

/***** Write a message *****/
static void doExpose(eventP)
XEvent *eventP;
{
    char message1[ ] = {"To scroll, press MB1."};
    char message2[ ] = {"To exit, click MB2."};

    XDrawImageString(dpy, win, gc, 150, 25, message1, strlen(message1));
    XDrawImageString(dpy, win, gc, 150, 50, message2, strlen(message2));
}

/***** Start a scroll operation *****/
static void startScroll()
{
❶  XCopyArea(dpy, win, win, gc, 0, scrollPixels,
        windowWidth, windowHeight, 0, 0);
    vY += scrollPixels;
}

/***** Copy the area *******/
static void doButtonPress(eventP)
XEvent *eventP;
{
    if (eventP->xbutton.button == Button2) sys$exit(1);
    ButtonIsDown = 1;
    startScroll();
    return;
}
```

(continued on next page)

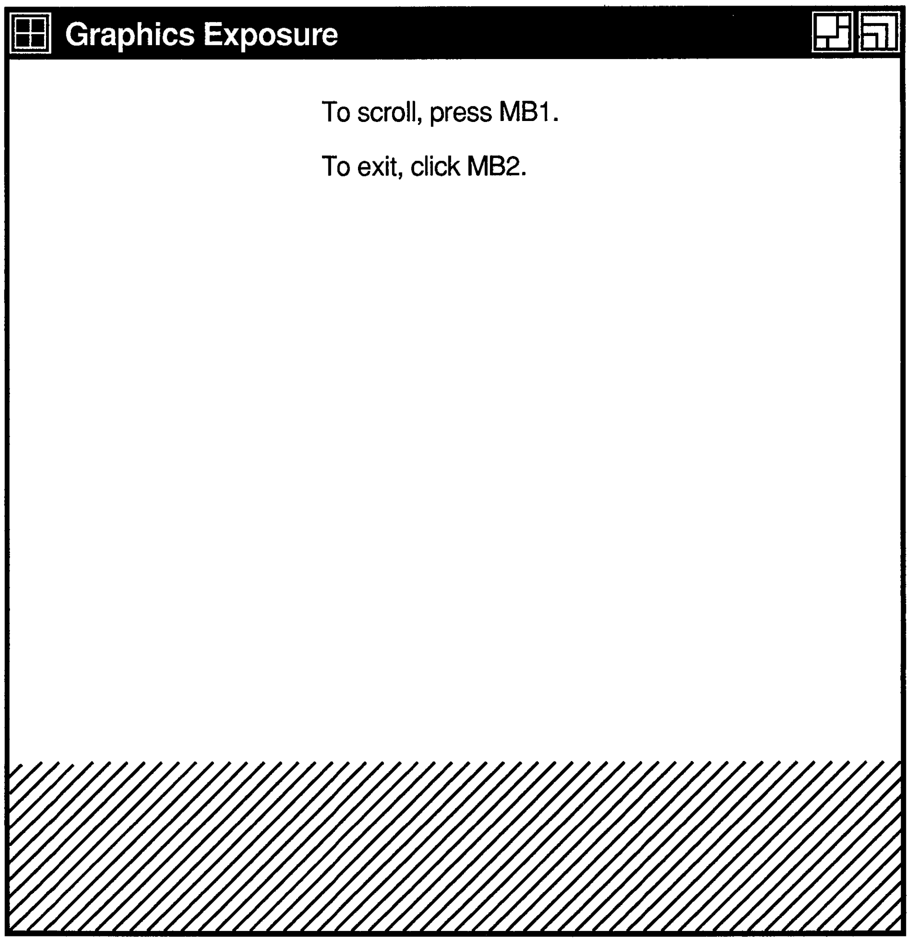
Example 9–5 (Cont.) Handling Graphics Exposures

```
/***** Draw points into the exposed area *******/
static void doGraphicsExpose(eventP)
XEvent *eventP;
{
❷   int x = eventP->xgraphicsexpose.x;
    int y = eventP->xgraphicsexpose.y;
    int width = eventP->xgraphicsexpose.width;
    int height = eventP->xgraphicsexpose.height;
    int px, py;

    for (py=y; py<(y+height); py++)
        for (px=x; px<(x+width); px++)
            if (!((px+py+vY) % 10)) XDrawPoint(dpy, win, gc, px, py);

    if (ButtonIsDown) startScroll();

    return;
}

/****** Quit scrolling when the button is released *******/
static void doButtonRelease(eventP)
XEvent *eventP;
{
    ButtonIsDown = 0;
    return;
}

/****** Draw points in the exposed area when window is obscured ****/
static void doNoExpose(eventP)
XEvent *eventP;
{
    if (ButtonIsDown) startScroll();
    return;
}
```

❶ The client-defined *startScroll* routine copies the window contents, less one row of pixels, to the top of the window. The result leaves an exposed area one pixel high at the bottom of the window.

❷ When a graphics exposure occurs, the client calculates where to draw points into the exposed area by referring to members of the expose event data structure.

Figure 9–2 Window Scrolling



ZK–0152A–GE

9.9 Window State Notification Events

Xlib reports events related to the state of a window when a client does one of the following:

- Circulates a window, changing the order of the window hierarchy
- Configures a window, changing its position, size, or border
- Creates a window
- Destroys a window
- Changes the size of a parent, causing Xlib to move a child window
- Maps a window
- Reparents a window

- Unmaps a window
- Changes the visibility of a window

This section describes handling events that result from these operations.

9.9.1 Handling Window Circulation

To receive notification when a client circulates a window, pass either the window identifier and the **StructureNotifyMask** mask or the identifier of the parent window and the **SubstructureNotifyMask** mask when using a selection method described in Section 9.2.

Xlib reports to interested clients a change in the hierarchical position of a window when a client calls the CIRCULATE SUBWINDOWS, CIRCULATE SUBWINDOWS UP, or CIRCULATE SUBWINDOWS DOWN routine.

Xlib uses the circulate event data structure to report circulate events. The following illustrates the data structure:

```
typedef struct {
        int type;
        unsigned long serial;
        Bool send_event;
        Display *display;
        Window event;
        Window window;
        int place;
} XCirculateEvent;
```

Table 9–19 describes members of the circulate event data structure.

Table 9–19 Circulate Event Data Structure Members

Member Name	Contents
type	Value defined by the CirculateNotify constant.
serial	Number of the last request processed by the server.
send_event	Value defined by the constant true if the event came from a SEND EVENT request.
display	Address of the display on which the event occurred.
event	Event window.
window	Window that has been circulated.
place	Place of the window on the stack after the window has been circulated. Xlib sets the value of this member to either the constant PlaceOnTop or the constant PlaceOnBottom. PlaceOnTop indicates that the window is above all siblings. PlaceOnBottom indicates that the window is below all siblings.

9.9.2 Handling Changes in Window Configuration

To receive notification when window size, position, border, or stacking order changes, pass either the window identifier and the **StructureNotifyMask** mask or the identifier of the parent window and the constant **SubstructureNotifyMask** when using the selection method described in Section 9.2.

Xlib reports changes in window configuration when the following occur:

- Window size, position, border, and stacking order change when a client calls the CONFIGURE WINDOW routine
- Window position in the stacking order changes when a client calls the LOWER WINDOW, RAISE WINDOW, or RESTACK WINDOW routine
- Window moves when a client calls the MOVE WINDOW routine
- Window size changes when a client calls the RESIZE WINDOW routine
- Window size and location change when a client calls the MOVE RESIZE WINDOW routine
- Border width changes when a client calls the SET WINDOW BORDER WIDTH routine

For more information about these routines, see Chapter 3.

Xlib reports changes to interested clients using the configure event data structure. The following illustrates the data structure:

```
typedef struct {
        int type;
        unsigned long serial;
        Bool send_event;
        Display *display;
        Window event;
        Window window;
        int x, y;
        int width, height;
        int border_width;
        Window above;
        Bool override_redirect;
} XConfigureEvent;
```

Table 9–20 describes members of the configure event data structure.

Table 9–20 Configure Event Data Structure Members

Member Name	Contents
type	Value defined by the ConfigureNotify constant.
serial	Number of the last request processed by the server.
send_event	Value defined by the constant true if the event came from a SEND EVENT request.

(continued on next page)

Table 9–20 (Cont.) Configure Event Data Structure Members

Member Name	Contents
display	Address of the display on which the event occurred.
event	Event window.
window	Window that has been reconfigured.
x	The x value of the coordinates that define the upper left corner of the window relative to the upper left corner of the parent window.
y	The y value of the coordinates that define the upper left corner of the window relative to the upper left corner of the parent window.
width	Width of the window, excluding the border.
height	Height of the window, excluding the border.
border_width	The width of the border in pixels.
above	The identifier of the sibling window above which the window is stacked. If this member has a value specified by the constant None, Xlib places the window at the bottom of the stack.
override_redirect	If true, this member specifies that the window manager ignore requests to reconfigure the window.

9.9.3 Handling Window Creations

To receive notification when a client creates a window, pass the identifier of the parent window and the constant **SubstructureNotifyMask** when using the selection method described in Section 9.2.

Xlib reports window creations using the create window event data structure. The following illustrates the data structure:

```
typedef struct {
        int type;
        unsigned long serial;
        Bool send_event;
        Display *display;
        Window parent;
        Window window;
        int x, y;
        int width, height;
        int border_width;
        Bool override_redirect;
} XCreateWindowEvent;
```

Table 9–21 describes members of the create window event data structure.

Table 9–21 Create Window Event Data Structure Members

Member Name	Contents
type	Value defined by either the FocusIn or the FocusOut constant.
serial	Number of the last request processed by the server.
send_event	Value defined by the constant true if the event came from a SEND EVENT request.
display	Address of the display on which the event occurred.
parent	Parent of the window created.
window	Window created.
x	The x value of the coordinates that define the origin of the window.
y	The y value of the coordinates that define the origin of the window.
width	Width of the window created.
height	Height of the window created.
border_width	Width of the border in pixels.
override_redirect	If true, this member specifies that the window manager ignore requests to create the window.

9.9.4 Handling Window Destructions

To receive notification when a client destroys a window, pass either the window identifier and the constant **StructureNotifyMask** or the identifier of the parent window and the **SubstructureNotifyMask** mask when using the selection method described in Section 9.2.

Xlib reports window destructions using the destroy window event data structure. The following illustrates the data structure:

```
typedef struct {
        int type;
        unsigned long serial;
        Bool send_event;
        Display *display;
        Window event;
        Window window;
} XDestroyWindowEvent;
```

Table 9–22 describes members of the destroy window event data structure.

Table 9–22 Destroy Window Event Data Structure Members

Member Name	Contents
type	Value defined by the DestroyNotify constant.
serial	Number of the last request processed by the server.

(continued on next page)

Table 9–22 (Cont.) Destroy Window Event Data Structure Members

Member Name	Contents
send_event	Value defined by the constant true if the event came from a SEND EVENT request.
display	Address of the display on which the event occurred.
event	Event window.
window	Window that has been destroyed.

9.9.5 Handling Changes in Window Position

To receive notification when a window is moved because a client has changed the size of its parent, pass the window identifier and the **StructureNotifyMask** mask or the identifier of the parent window and the **SubstructureNotifyMask** mask when using the selection method described in Section 9.2.

Xlib reports window gravity events using the gravity event data structure. The following illustrates the data structure:

```
typedef struct {
        int type;
        unsigned long serial;
        Bool send_event;
        Display *display;
        Window event;
        Window window;
        int x, y;
} XGravityEvent;
```

Table 9–23 describes members of the gravity event data structure.

Table 9–23 Gravity Event Data Structure Members

Member Name	Contents
type	Value defined by the GravityNotify constant.
serial	Number of the last request processed by the server.
send_event	Value defined by the constant true if the event came from a SEND EVENT request.
display	Address of the display on which the event occurred.
event	Event window.
window	Window that has been moved.
x	The x value of the coordinates that define the upper left corner of the window relative to the upper left corner of the parent window.
y	The y value of the coordinates that define the upper left corner of the window relative to the upper left corner of the parent window.

9.9.6 Handling Window Mappings

To receive notification when a window changes state from unmapped to mapped, pass either the window identifier and the **StructureNotifyMask** mask or the identifier of the parent window and the **SubstructureNotifyMask** mask when using the selection method described in Section 9.2.

Xlib reports window mapping events using the map event data structure. The following illustrates the data structure:

```
typedef struct {
        int type;
        unsigned long serial;
        Bool send_event;
        Display *display;
        Window event;
        Window window;
        Bool override_redirect;
} XMapEvent;
```

Table 9–24 describes members of the map event data structure.

Table 9–24 Map Event Data Structure Members

Member Name	Contents
type	Value defined by the MapNotify constant.
serial	Number of the last request processed by the server.
send_event	Value defined by the constant true if the event came from a SEND EVENT request.
display	Address of the display on which the event occurred.
event	Event window.
window	Window that has been mapped.
override_redirect	If true, indicates that the window manager should disregard requests to map the window. When true, it overrides a substructure redirect on the parent.

9.9.7 Handling Key, Keyboard, and Pointer Mappings

All clients receive notification of changes in key, keyboard, and pointer mapping. Xlib reports these events when a client has successfully done one of the following:

- Called the SET MODIFIER MAPPING routine to indicate which keycodes are modifiers
- Changed keyboard mapping using the CHANGE KEYBOARD MAPPING routine
- Set pointer mapping using the SET POINTER MAPPING routine

Xlib reports key, keyboard, and pointer mapping events using the mapping event data structure. The following illustrates the data structure:

```
typedef struct {
        int type;
        unsigned long serial;
        Bool send_event;
        Display *display;
        Window window;
        int request;
        int first_keycode;
        int count;
} XMappingEvent;
```

Table 9–25 describes members of the mapping event data structure.

Table 9–25 Mapping Event Data Structure Members

Member Name	Contents	
type	Value defined by the MappingNotify constant.	
serial	Number of the last request processed by the server.	
send_event	Value defined by the constant true if the event came from a SEND EVENT request.	
display	Address of the display on which the event occurred.	
event	Event window.	
window	Unused member.	
request	The type of mapping change being reported. Possible values are indicated by the following constants:	
	MappingModifier	Specified key codes are used as modifiers.
	MappingKeyboard	Keyboard mapping has changed. Sets the first_ keycode and count members.
	MappingPointer	Pointer button mapping is set.
first_keycode	First number of the range of altered keys, set only if the request member has a value specified by the constant MappingKeyboard.	
count	Last number of the range of altered keys, set only if the request member has a value specified by the constant MappingKeyboard.	

9.9.8 Handling Window Reparenting

To receive notification when the parent of a window changes, pass either the window identifier and the **StructureNotifyMask** mask or the identifier of the parent window and the **SubstructureNotifyMask** mask when using the selection method described in Section 9.2.

Xlib reports window reparenting events using the reparent event data structure. The following illustrates the data structure:

```
typedef struct {
        int type;
        unsigned long serial;
        Bool send_event;
        Display *display;
        Window event;
        Window window;
        Window parent;
        int x, y;
        Bool override_redirect;
} XReparentEvent;
```

Table 9–26 describes members of the reparent event data structure.

Table 9–26 Reparent Event Data Structure Members

Member Name	Contents
type	Value defined by the ReparentNotify constant.
serial	Number of the last request processed by the server.
send_event	Value defined by the constant true if the event came from a SEND EVENT request.
display	Address of the display on which the event occurred.
event	Event window.
window	Window reparented.
parent	New parent of the window.
x	The x value of the coordinates that define the upper left corner of the window relative to the upper left corner of the parent window.
y	The y value of the coordinates that define the upper left corner of the window relative to the upper left corner of the parent window.
override_redirect	If true, this member specifies that the window manager ignore requests to reparent the window. When true, it overrides a substructure redirect on the parent.

9.9.9 Handling Window Unmappings

To receive notification when a window changes from mapped to unmapped, pass either the window identifier and the **StructureNotifyMask** mask or the identifier of the parent window and the **SubstructureNotifyMask** mask when using the selection method described in Section 9.2.

Xlib reports window unmapping events using the unmap event data structure. The following illustrates the data structure:

```
typedef struct {
        int type;
        unsigned long serial;
        Bool send_event;
        Display *display;
        Window event;
        Window window;
        Bool from_configure;
} XUnmapEvent;
```

Table 9–27 describes members of the unmap event data structure.

Table 9–27 Unmap Event Data Structure Members

Member Name	Contents
type	Value defined by the UnmapNotify constant.
serial	Number of the last request processed by the server.
send_event	Value defined by the constant true if the event came from a SEND EVENT request.
display	Address of the display on which the event occurred.
event	Event window.
window	Window unmapped.
from_configure	If true, indicates that the event occurred as a result of resizing the parent window when the window itself has a window gravity specified by the constant UnmapGravity.

9.9.10 Handling Changes in Window Visibility

All or part of a window is visible if it is mapped to a screen, if all of its ancestors are mapped, and if it is at least partially visible on the screen. To receive notification when the visibility of a window changes, pass the window identifier and the **StructureNotifyMask** mask when using the selection method described in Section 9.2.

Xlib reports changes in visibility to interested clients using the visibility event data structure. The following illustrates the data structure:

```
typedef struct {
        int type;
        unsigned long serial;
        Bool send_event;
        Display *display;
        Window window;
        int state;
} XVisibilityEvent;
```

Table 9–28 describes members of the visibility event data structure.

Table 9–28 Visibility Event Data Structure Members

Member Name	Contents
type	Value defined by the VisibilityNotify constant.
serial	Number of the last request processed by the server.
send_event	Value defined by the constant true if the event came from a SEND EVENT request.

(continued on next page)

Table 9–28 (Cont.) Visibility Event Data Structure Members

Member Name	Contents
display	Address of the display on which the event occurred.
window	Window whose visibility changed.
state	If set to the value defined by the VisibilityUnobscured constant, the window has changed from being partially and fully obscured to being fully visible. If set to the value defined by the VisibilityPartiallyObscured, the window has changed from being fully obscured or fully visible to partially obscured. If set to the value defined by the VisibilityFullyObscured constant, the window has changed from being fully visible or partially obscured to not visible.

9.10 Color Map State Events

Xlib reports a color map event when the window manager installs, changes, or removes the color map.

To receive notification of color map events, pass the window identifier and the **ColormapChangeMask** mask when using the selection method described in Section 9.2.

Xlib reports color map events to interested clients when the following occur:

- A client sets the color map member of the set window attributes data structure by calling CHANGE WINDOW ATTRIBUTES. See Chapter 3 for more information on the data structure and routine.
- A client calls the FREE COLORMAP routine. See Section 5.5 for more information about FREE COLORMAP.
- The window manager installs or removes a color map in response to either a client call of the INSTALL COLORMAP or UNINSTALL COLORMAP routine.

Xlib reports color map events using the color map event data structure. The following illustrates the data structure:

```
typedef struct {
        int type;
        unsigned long serial;
        Bool send_event;
        Display *display;
        Window window;
        Colormap colormap;
        Bool new;
        int state;
} XColormapEvent;
```

Table 9–29 describes members of the color map event data structure.

Table 9–29 Color Map Event Data Structure Members

Member Name	Contents
type	Value defined by the ColormapNotify constant.
serial	Number of the last request processed by the server.
send_event	Value defined by the constant true if the event came from a SEND EVENT request.
display	Address of the display on which the event occurred.
window	Window whose associated color map has changed.
colormap	If the window manager changes the color map in response to a call to CHANGE WINDOW ATTRIBUTES, INSTALL COLORMAP, or UNINSTALL COLORMAP, this member has a value specified by the constant Colormap. If the window manager changes the color map in response to a call to FREE COLORMAP, this member has a value specified by the constant None.
new	Value defined by the constant true if the window manager has changed the color map. The value defined by the constant false if the window manager has installed or removed the color map.
state	Value defined by the constant ColormapInstalled if the color map is installed. The value defined by the constant ColormapUninstalled if the color map is not installed.

9.11 Client Communication Events

Xlib reports an event when one of the following occurs:

- One client notifies another client that an event has happened.
- A client changes, deletes, rotates, or gets a property.
- A client loses ownership of a window.
- A client requests ownership of a window.

This section describes how to handle communication between clients.

9.11.1 Handling Event Notification from Other Clients

Clients can notify each other of events by calling the SEND EVENT routine.

Xlib sends notification between clients using the client message event data structure. The following illustrates the data structure:

```
typedef struct {
        int type;
        unsigned long serial;
        Bool send_event;
        Display *display;
        Window window;
        Atom message_type;
        int format;
        union {
                char b[20];
                short s[10];
                int l[5];
                } data;
} XClientMessageEvent;
```

Table 9–30 describes members of the client message event data structure.

Table 9–30 Client Message Event Data Structure Members

Member Name	Contents
type	Value defined by the ClientMessage constant.
serial	Number of the last request processed by the server.
send_event	Value defined by the constant true if the event came from a SEND EVENT request.
display	Address of the display on which the event occurred.
window	Window to which the message is sent.
message_type	Indicates how the message data is to be interpreted by the receiving client. For more information about atoms, see Chapter 3.
format	Indicates whether the data is in units of 8, 16, or 32 bits.
b	Data of 20 8-bit values.
s	Data of 10 16-bit values.
l	Data of 5 32-bit values.

9.11.2 Handling Changes in Properties

As Chapter 3 notes, a property associates a constant with data of a particular type. Xlib reports a property event when a client does one of the following:

- Changes a property
- Rotates a window property
- Gets a property
- Deletes a property

To receive information about property changes, pass the window identifier and the **PropertyChangeMask** mask when using the selection method described in Section 9.2.

Xlib reports changes in properties to interested clients using the property event data structure. The following illustrates the data structure:

```
typedef struct {
        int type;
        unsigned long serial;
        Bool send_event;
        Display *display;
        Window window;
        Atom atom;
        Time time;
        int state;
} XPropertyEvent;
```

Table 9–31 describes members of the property event data structure.

Table 9–31 Property Event Data Structure Members

Member Name	Contents
type	Value defined by the PropertyNotify constant.
serial	Number of the last request processed by the server.
send_event	Value defined by the constant true if the event came from a SEND EVENT request.
display	Address of the display on which the event occurred.
window	Window whose property was changed.
atom	Identifies the property that was changed. For more information about properties and atoms, see Chapter 3.
time	Server time that the property changed.
state	Value specified by the constant PropertyNewValue if a client changes a property by calling either the CHANGE PROPERTY or the ROTATE PROPERTY routine. The same result occurs if the client replaces all or part of a property with identical data using CHANGE PROPERTY or ROTATE PROPERTY.
	The value specified by the constant PropertyDelete if a client deletes a property by calling either the DELETE PROPERTY or the GET PROPERTY routine. For more information about properties, see Chapter 3.

9.11.3 Handling Changes in Selection Ownership

Clients receive notification automatically when they lose ownership of a selection in a window. Xlib reports the event when a client takes ownership of a selection by calling the SET SELECTION OWNER routine.

To report the event, Xlib uses the selection clear event data structure. The following illustrates the data structure:

```
typedef struct {
        int type;
        unsigned long serial;
        Bool send_event;
        Display *display;
        Window window;
        Atom selection;
        Time time;
} XSelectionClearEvent;
```

Table 9–32 describes members of the selection clear event data structure.

Table 9–32 Selection Clear Event Data Structure Members

Member Name	Contents
type	Value defined by the SelectionClear constant.
serial	Number of the last request processed by the server.
send_event	Value defined by the constant true if the event came from a SEND EVENT request.
display	Address of the display on which the event occurred.
window	Window losing ownership of the selection.
selection	Selection atom. For more information about atoms and selection, see Chapter 3.
time	Last time change recorded for the selection.

9.11.4 Handling Requests to Convert a Selection

The server issues a selection request event to the owner of a selection when a client calls the CONVERT SELECTION routine. For information about the CONVERT SELECTION routine, see Section 3.5.2.

To report the event, Xlib uses the selection request event data structure. The following illustrates the data structure:

```
typedef struct {
        int type;
        unsigned long serial;
        Bool send_event;
        Display *display;
        Window owner;
        Window requestor;
        Atom selection;
        Atom target;
        Atom property;
        Time time;
} XSelectionRequestEvent;
```

Table 9–33 describes members of the selection request event data structure.

Table 9–33 Selection Request Event Data Structure Members

Member Name	Contents
type	Value defined by the SelectionRequest constant.
serial	Number of the last request processed by the server.
send_event	Value defined by the constant true if the event came from a SEND EVENT request.
display	Address of the display on which the event occurred.
owner	Window that owns the selection.
requestor	Window that requests the selection.
selection	Selection atom. For more information about atoms and selection, see Chapter 3.
target	Data type that selection is converted to before being returned.
property	Atom that specifies a property or the constant None.
time	Timestamp, expressed in milliseconds, or the constant CurrentTime from the convert selection request

9.11.5 Handling Requests to Notify of a Selection

The server issues a selection notify event after a client calls the CONVERT SELECTION routine. The owner of the selection being converted should initiate this event by calling SEND EVENT when either the selection has been converted and stored as a property or the selection conversion could not be performed. For information about converting selections, see Section 3.5.2.

To report the event, Xlib uses the selection event data structure. The following illustrates the data structure:

```
typedef struct {
        int type;
        unsigned long serial;
        Bool send_event;
        Display *display;
        Window requestor;
        Atom selection;
        Atom target;
        Atom property;
        Time time;
} XSelectionEvent;
```

Table 9–34 describes members of the selection event data structure.

Table 9–34 Selection Event Data Structure Members

Member Name	Contents
type	Value defined by the SelectionNotify constant.
serial	Number of the last request processed by the server.
send_event	Value defined by the constant true if the event came from a SEND EVENT request.
display	Address of the display on which the event occurred.
requestor	Window that requests the selection.
selection	Selection atom. For more information about atoms and selection, see Chapter 3.
target	Data type to which selection is converted.
property	Atom that specifies a property or the constant None.
time	Timestamp, expressed in milliseconds, or the constant CurrentTime from the convert selection request.

9.12 Event Queue Management

Xlib maintains an input queue known as the **event queue**. When an event occurs, the server sends the event to Xlib, which places it at the end of an event queue. By using routines described in this section, the client can check, remove, and process the events on the queue. As the client removes an event, remaining events move up the event queue.

Certain routines may **block** or prevent other routine calls from accessing the event queue. If the blocking routine does not find an event that the client is interested in, Xlib flushes the output buffer and waits until an event is received from the server.

This section describes how the event queue is managed, including the following topics:

- Checking events on the queue
- Returning events in order and removing them from the queue
- Returning events without removing them from the queue
- Obtaining events that match the event mask or the arbitrary functions that the client provides
- Putting events back onto the event queue
- Sending events to other clients

9.12.1 Checking the Contents of the Event Queue

To check the event queue without preventing other routines from accessing the queue, use the EVENTS QUEUED routine. Clients can check events already queued by calling the EVENTS QUEUED routine and specifying one of the following constants:

QueuedAlready	Returns the number of events already in the event queue and never performs a system call.
QueuedAfterFlush	Returns the number of events in the event queue if the value is a nonzero. If there are no events in the queue, this routine flushes the output buffer, attempts to read more events out of the client connection, and returns the number read.
QueuedAfterReading	Returns the number of events already in the event queue if the value is a nonzero. If there are no events in the queue, this routine attempts to read more events out of the client connection without flushing the output buffer and returns the number read.

To return the number of events in the event queue, use the PENDING routine. If there are no events in the queue, PENDING flushes the output buffer, attempts to read more events out of the client connection, and returns the number read. The PENDING routine is identical to EVENTS QUEUED with constant **QueuedAfterFlush** specified.

9.12.2 Returning the Next Event on the Queue

To return the first event on the event queue and copy it into the specified event data structure, use the NEXT EVENT and PEEK EVENT routines. NEXT EVENT returns the first event, copies it into an EVENT structure, and removes it from the queue. PEEK EVENT returns the first event, copies it into an event data structure, but does not remove it from the queue. In both cases, if the event queue is empty, the routine flushes the output buffer and blocks until an event is received.

9.12.3 Selecting Events That Match User-Defined Routines

Xlib enables the client to check all the events on the queue for a specific type of event by specifying a client-defined routine known as a **predicate procedure**. The predicate procedure determines if the event on the queue is one that the client is interested in.

The client calls the predicate procedure from inside the event routine. The predicate procedure should determine only if the event is useful and must not call Xlib routines. The predicate procedure is called once for each event in the queue until it finds a match.

Table 9–35 lists routines that use a predicate procedure and indicates whether or not the routine blocks.

Table 9–35 Selecting Events Using a Predicate Procedure

Routine	Description	Blocking/No Blocking
IF EVENT	Checks the event queue for the specified event. If the event matches, removes the event from the queue. This routine is also called each time an event is added to the queue.	Blocking
CHECK IF EVENT	Checks the event queue for the specified event. If the event matches, removes the event from the queue. If the predicate procedure does not find a match, it flushes the output buffer.	No blocking
PEEK IF EVENT	Checks the event queue for the specified event but does not remove it from the queue. This routine is also called each time an event is added to the queue.	Blocking

9.12.4 Selecting Events Using an Event Mask

Xlib enables a client to process events out of order by specifying a window identifier and one of the event masks listed in Table 9–3 when calling routines listed in Table 9–36.

For example, the following specifies keyboard events on window *window* by using the event mask name constant **KeymapStateMask**.

```
     .
     .
     .
XWindowEvent(dpy, window, KeymapState, &event)
```

Table 9–36 lists routines that use event or window masks and indicates whether the routine blocks.

Table 9–36 Routines to Select Events Using a Mask

Routine	Description	Blocking/No Blocking
WINDOW EVENT	Searches the event queue and removes the next event that matches both the specified window and event mask	Blocking
CHECK WINDOW EVENT	Searches the event queue, then the events available on the server connection, and removes the first event that matches the specified event and window mask	No blocking
MASK EVENT	Searches the event queue and removes the next event that matches the event mask	Blocking

(continued on next page)

Table 9–36 (Cont.) Routines to Select Events Using a Mask

Routine	Description	Blocking/No Blocking
CHECK MASK EVENT	Searches the event queue, then the events available on the server connection, and removes the next event that matches an event mask	No blocking
CHECK TYPED EVENT	Returns the next event in the queue that matches an event type	No blocking
CHECK TYPED WINDOW EVENT	Searches the event queue, then the events available on the server connection, and removes the next event that matches the specified type and window	No blocking

9.12.5 Putting an Event Back on Top of the Queue

To push an event back onto the top of the event queue, use the PUT BACK EVENT routine. PUT BACK EVENT is useful when a client returns an event from the queue and decides to use it later. There is no limit to how many times in succession PUT BACK EVENT can be called.

9.12.6 Sending Events to Other Clients

To send an event to a client, use the SEND EVENT routine. For example, owners of a selection should use this routine to send a SELECTION NOTIFY event to a requestor when a selection has been converted and stored as a property.

9.13 Error Handling

Xlib has two default error handlers. One manages fatal errors, such as when the connection to a display is severed due to a system failure. The other handles error events from the server. The default error handlers print an explanatory message and text and then exit.

Each of these error handlers can be replaced by client error handling routines. If a client-supplied routine is passed a null pointer, Xlib reinvokes the default error handler.

This section describes the Xlib event error handling resources including enabling synchronous operation, handling server errors, and handling input/output (I/O) errors.

9.13.1 Enabling Synchronous Operation

When debugging programs it is convenient to require Xlib to behave synchronously so that errors are reported at the time they occur.

To enable synchronous operation, use the SYNCHRONIZE routine. The client passes the **display** argument and the **onoff** argument. The **onoff** argument passes either a value of zero (disabling synchronization) or a nonzero value (enabling synchronization).

9.13.2 Using the Default Error Handlers

To handle error events when an error event is received, use the SET ERROR HANDLER routine.

Xlib provides an error event data structure that passes information to the SET ERROR HANDLER routine.

The following illustrates the error event data structure:

```
typedef struct {
        int type
        Display *display
        unsigned long serial;
        char error_code;
        char request_code;
        char minor_code;
        XID resourceid;
} XErrorEvent;
```

Table 9–37 describes the members of the data structure.

Table 9–37 Error Event Data Structure Members

Member Name	Description
type	Type of error event being reported
display	Display on which the error event occurred
serial	Number of requests starting at one sent over the network connection since it was opened
error_code	Identifying error code of the failing routine
request_code	Protocol representation of the name of the procedure that failed and defined in X11/X.h
minor_code	Minor opcode of failed request
resourceid	Resource ID

The routines described in this section return Xlib error codes. Table 9–38 lists the codes and describes the errors.

Table 9–38 Event Error Codes

Error Code	Description
BadAccess	Possible causes are: • An attempt to grab a key/button combination that has already been grabbed by another client. • An attempt to free a color map entry that was not allocated by the client. • An attempt to store into a read-only, or unallocated, color map entry. • An attempt to modify the access control list from other than the local host. • An attempt to select an event type that only one client can select at a time when another client has already selected it.
BadAlloc	The server did not allocate the requested resource for any cause.
BadAtom	The value specified in an atom argument does not name a defined atom.
BadColor	A value specified for a color map argument does not name a defined color map.
BadCursor	A value specified for a cursor argument does not name a defined cursor.
BadDrawable	A value specified for a drawable argument does not name a defined window or pixmap.
BadFont	A value specified for a font argument does not name a defined font (or, in some cases, graphics context).
BadGC	A value specified for a graphics context argument does not name a defined graphics context.
BadIDChoice	The value specified for a resource identifier is either not included in the range assigned to the client, or it is already in use. Under normal circumstances this cannot occur and should be considered a server or Xlib error.
BadImplementation	The server does not implement some aspect of the request. This error is most likely caused by a server extension; a server that generates this error for a core protocol request is deficient. As such, this error is not listed for any particular request. Clients should be prepared to receive this type of error and either handle or discard it.
BadLength	The length of a request is shorter or longer than required to minimally contain the arguments. This error usually indicates an internal Xlib or server error. The length of a request exceeds the maximum length accepted by the server.

(continued on next page)

Table 9–38 (Cont.) Event Error Codes

Error Code	Description
BadMatch	Possible causes are: • In a graphics request, the root and depth of the graphics context does not match that of the drawable. • An input-only window is used as a drawable. • One argument or pair of arguments has the correct type and range but fails to match in some other way required by the request. • An input only window lacks this attribute.
BadName	The font or color specified does not exist.
BadPixmap	A value specified for a pixmap argument does not name a defined pixmap.
BadRequest	The major or minor opcode specified does not indicate a valid request. This is usually an Xlib or server error.
BadValue	Some numeric values fall outside the range of values accepted by the request. Unless a specific range is specified for an argument, the full range defined by the argument's type is accepted. Any argument defined as a set of alternatives can generate this error.
BadWindow	A value specified for a window argument does not name a defined window.

Note that Bad Atom, Bad Color, Bad Cursor, Bad Drawable, Bad Font, Bad Pixmap, and Bad Window errors are also used when the argument type is extended by a set of fixed alternatives.

To obtain a text description of the specified error code, use the GET ERROR TEXT routine. This routine copies a null terminated string describing the specified error code into the specified buffer. The client should use this routine to obtain an error description because extensions to Xlib may define their own error codes and error strings.

To obtain error messages from the error database, use the GET ERROR DATABASE TEXT routine. This routine returns a message (or the default message) from the error message database. The GET ERROR DATABASE TEXT uses the resource manager to look up a string and returns it in the buffer argument. Xlib uses this function internally to look up its error messages.

To report an error when the requested display does not exist, use the DISPLAY NAME routine. This routine returns the name of the display that the client is currently using. The DISPLAY NAME routine passes the argument **string**. If null string is specified, DISPLAY NAME looks in the environment and returns the display name requested. This makes it easier to report precisely which display the client attempted to open when the initial connection attempt failed.

To handle fatal I/O errors, use the SET IO ERROR HANDLER routine. Xlib calls the supplied error handler if any system call error occurs (for example, the connection to the server is lost). In this case, the called routine should not return. If the I/O handler does return, the client exits.

A Compiling Fonts

VMS DECwindows includes a font compiler that enables programmers to convert an ASCII bitmap distribution format (BDF) into binary server natural form (SNF). The server uses an SNF file to display a font. In addition to converting the BDF file to binary form, the compiler provides statistical information about the font and the compilation process.

To invoke the font compiler, use the following format:

```
FONT filespec [
               /[NO]OUTPUT[=filename]
               /[NO]MINBBOX
               /[NO]REPORT
               ]
```

The **filename** parameter specifies the BDF file to be converted. A file name is required. The default value of the optional file type is **DECW$BDF**.

The /OUTPUT qualifier specifies the file name and type of the resulting SNF file. The default output file name is the file name of the BDF file being converted. The default output SNF file type is **DECW$FONT**.

Compiler output consists of a header file that contains font information, character metrics, and the image of each character in the font. Font information in the header file is essentially the same as information stored in the font struct data structure. For a description of the data structure, see Section 8.1.

The /MINBBOX qualifier specifies that the compiler produce the minimum bounding box for each character in the font and adjust values for the left bearing, right bearing, ascent, and descent of each character accordingly. Character width is not affected. Specifying the /MINBBOX qualifier is equivalent to converting a fixed font to a monospaced font. For a description of character metrics and fonts, see Section 8.1.

Using the /MINBBOX qualifier has two advantages. Because the font compiler produces minimum instead of fixed bounding boxes, the resulting SNF file is significantly smaller than the comparable fixed font SNF file. Consequently, both disk requirements for storing the font and server memory requirements when a client loads the font are reduced. Also, because the resulting font comprises minimum inkable characters, server performance when writing text is increased as much as 20 percent.

The /REPORT qualifier directs the compiler to report information about the font and the conversion process, including BDF information, font properties, compiler generation information, and metrics. The /REPORT qualifier also causes the compiler to illustrate each glyph in the font.

B Routines Requiring Protocol Requests

Table B–1 lists Xlib routines requiring protocol requests. The table provides the protocol request and a short description for each Xlib function.

Table B–1 Routines Requiring Protocol Requests

Xlib Function	Protocol Request	Description
ALLOC COLOR	ALLOC COLOR	Allocates a read-only color cell
ALLOC COLOR CELLS	ALLOC COLOR CELLS	Allocates read/write color cells and color plane combinations for a PseudoColor model
ALLOC COLOR PLANES	ALLOC COLOR PLANES	Allocates read/write color resources for DirectColor visual types
ALLOC NAMED COLOR	ALLOC NAME COLOR	Allocates a read-only color cell by name and returns the closest color supported by the hardware
CHANGE GC	CHANGE GC	Changes the components in the specified graphics context
CHANGE PROPERTY	CHANGE PROPERTY	Changes the property of a specified window
CHANGE WINDOW ATTRIBUTES	CHANGE WINDOW ATTRIBUTES	Changes one or more window attributes
CIRCULATE SUBWINDOWS	CIRCULATE WINDOW	Circulates a subwindow up or down
CIRCULATE SUBWINDOWS DOWN	CIRCULATE WINDOW	Lowers the highest mapped child of a window that partially or completely occludes another child
CIRCULATE SUBWINDOWS UP	CIRCULATE WINDOW	Raises the lowest mapped child of an occluded window
CLEAR AREA	CLEAR AREA	Clears a specified rectangular area of the specified window
CLEAR WINDOW	CLEAR AREA	Clears the entire area in the specified window
CONFIGURE WINDOW	CONFIGURE WINDOW	Configures a window's size, location, stacking, or border
CONVERT SELECTION	CONVERT SELECTION	Requests conversion of a selection
COPY AREA	COPY AREA	Copies an area of the specified drawable between drawables of the same root and depth

(continued on next page)

Table B–1 (Cont.) Routines Requiring Protocol Requests

Xlib Function	Protocol Request	Description
COPY COLORMAP AND FREE	COPY COLORMAP AND FREE	Creates a new color map when allocating out of a previously shared color map has failed due to resource exhaustion
COPY GC	COPY GC	Copies components from a source graphics context to a destination graphics context
COPY PLANE	COPY PLANE	Copies a single bit-plane of the specified drawable
CREATE COLORMAP	CREATE COLORMAP	Creates a color map for a screen
CREATE FONT CURSOR	CREATE GLYPH CURSOR	Creates a cursor from a standard font
CREATE GC	CREATE GC	Creates a new graphics context that is usable with the specified drawable
CREATE GLYPH CURSOR	CREATE GLYPH CURSOR	Creates a cursor from font glyphs
CREATE PIXMAP	CREATE PIXMAP	Creates a pixmap of a specified size
CREATE PIXMAP CURSOR	CREATE CURSOR	Creates a cursor from two bitmaps
CREATE SIMPLE WINDOW	CREATE WINDOW	Creates an unmapped input-output subwindow of the specified parent window
CREATE WINDOW	CREATE WINDOW	Creates an unmapped subwindow for a specified parent window
DEFINE CURSOR	CHANGE WINDOW ATTRIBUTES	Defines which cursor will be used in a window
DELETE PROPERTY	DELETE PROPERTY	Deletes a property for the specified window
DESTROY SUBWINDOWS	DESTROY SUBWINDOWS	Destroys all subwindows of a specified window
DESTROY WINDOW	DESTROY WINDOW	Destroys a window and all of its subwindows
DRAW ARC	POLY ARC	Draws a single arc in the specified drawable
DRAW ARCS	POLY ARC	Draws multiple arcs in the specified drawable
DRAW IMAGE STRING	IMAGE TEXT 8	Draws 8-bit image text characters in the specified drawable
DRAW IMAGE STRING 16	IMAGE TEXT 16	Draws 2-byte image text characters in the specified drawable
DRAW LINE	POLY SEGMENT	Draws a single line between two points in the specified drawable
DRAW LINES	POLY LINE	Draws multiple lines in the specified drawable

(continued on next page)

Table B–1 (Cont.) Routines Requiring Protocol Requests

Xlib Function	Protocol Request	Description
DRAW POINT	POLY POINT	Draws a single point in the specified drawable
DRAW POINTS	POLY POINT	Draws multiple points in the specified drawable
DRAW RECTANGLE	POLY RECTANGLE	Draws the outline of a single rectangle in the specified drawable
DRAW RECTANGLES	POLY RECTANGLE	Draws the outline of multiple rectangles in the specified drawable
DRAW SEGMENTS	POLY SEGMENT	Draws multiple but not necessarily connected lines in the specified drawable
DRAW STRING	POLY TEXT 8	Draws 8-bit characters in the specified drawable
DRAW STRING 16	POLY TEXT 16	Draws 2-byte characters in the specified drawable
DRAW TEXT	POLY TEXT 8	Draws 8-bit characters in the specified drawable
DRAW TEXT 16	POLY TEXT 16	Draws 2-byte characters in the specified drawable
FETCH BYTES	GET PROPERTY	Returns data from cut buffer 0
FETCH NAME	GET PROPERTY	Gets the name of a window
FILL ARC	POLY FILL ARC	Fills a single arc in the specified drawable
FILL ARCS	POLY FILL ARC	Fills multiple arcs in the specified drawable
FILL POLYGON	FILL POLY	Fills a polygon area in the specified drawable
FILL RECTANGLE	POLY FILL RECTANGLE	Fills a single rectangular area in the specified drawable
FILL RECTANGLES	POLY FILL RECTANGLE	Fills multiple rectangular areas in the specified drawable
FREE COLORMAP	FREE COLOR MAP	Deletes the association between the color map resource ID and the color map
FREE COLORS	FREE COLOR	Frees color map cells
FREE CURSOR	FREE CURSOR	Frees (destroys) the specified cursor
FREE FONT	CLOSE FONT	Unloads the font and frees the storage used by the font data structure that was allocated by QUERY FONT and LOAD QUERY FONT
FREE GC	FREE GC	Frees the specified graphics context

(continued on next page)

Table B–1 (Cont.) Routines Requiring Protocol Requests

Xlib Function	Protocol Request	Description
FREE PIXMAP	FREE PIXMAP	Frees all storage associated with a specified pixmap
GET ATOM NAME	GET ATOM NAME	Returns a name for the specified atom identifier
GET FONT PATH	GET FONT PATH	Gets the current font search path
GET GEOMETRY	GET GEOMETRY	Obtains the current geometry of the specified drawable
GET ICON SIZES	GET PROPERTY	Returns the value of the icon sizes atom
GET IMAGE	GET IMAGE	Returns the contents of a rectangle in the specified drawable on the display
GET MOTION EVENTS	GET MOTION EVENTS	Gets the motion history for a specified window and time
GET NORMAL HINTS	GET PROPERTY	Returns the size hints for a window in its normal state
GET SELECTION OWNER	GET SELECTION OWNER	Returns the selection owner
GET SIZE HINTS	GET PROPERTY	Reads the value of any property of type WM_SIZE_HINTS
GET WM HINTS	GET PROPERTY	Reads the value of the window manager hints atom
GET WINDOW ATTRIBUTES	GET WINDOW ATTRIBUTES GET GEOMETRY	Obtains the current attributes or geometry of a specified window
GET WINDOW PROPERTY	GET PROPERTY	Obtains the atom type and property format of a specified window
GET ZOOM HINTS	GET PROPERTY	Reads the value of the zoom hints atom
INIT EXTENSION	QUERY EXTENSION	Allocates storage for maintaining the information about the extension on the connection, chains this onto the extension list, and returns the information the stub implementor needs to access the extension
INTERN ATOM	INTERN ATOM	Returns an atom for a specified name
LIST EXTENSIONS	LIST EXTENSIONS	Returns a list of all extensions supported by the server
LIST FONTS	LIST FONTS	Returns a list of the available font names
LIST FONTS WITH INFO	LIST FONTS WITH INFO	Obtains the names and information about loaded fonts
LIST PROPERTIES	LIST PROPERTIES	Obtains the specified window's property list
LOAD FONT	OPEN FONT	Loads the specified font

(continued on next page)

Table B–1 (Cont.) Routines Requiring Protocol Requests

Xlib Function	Protocol Request	Description
LOAD QUERY FONT	OPEN FONT QUERY FONT	Performs a LOAD FONT and QUERY FONT in a single operation
LOOKUP COLOR	LOOKUP COLOR	Looks up the name of a color
LOWER WINDOW	CONFIGURE WINDOW	Lowers a window so that it does not obscure any sibling window
MAP RAISED	CONFIGURE WINDOW MAP WINDOW	Maps and raises a window
MAP SUBWINDOWS	MAP SUBWINDOWS	Maps all subwindows for a specified window
MAP WINDOW	MAP WINDOW	Maps the specified window
MOVE RESIZE WINDOW	CONFIGURE WINDOW	Changes size and location of a window
MOVE WINDOW	CONFIGURE WINDOW	Moves a window without changing its size
NO OP	NO OPERATION	Sends a NoOperation request to the server
OPEN DISPLAY	CREATE GC	Opens a connection to the server controlling the specified display
PARSE COLOR	LOOKUP COLOR	Parses color values
PUT IMAGE	PUT IMAGE	Combines an image in memory with a rectangle of a drawable on the display
QUERY BEST CURSOR	QUERY BEST SIZE	Determines useful cursor sizes
QUERY BEST SIZE	QUERY BEST SIZE	Obtains the best size of a tile, stipple, or cursor
QUERY BEST STIPPLE	QUERY BEST SIZE	Obtains the best stipple shape
QUERY BEST TILE	QUERY BEST SIZE	Obtains the fill tile shape
QUERY COLOR	QUERY COLORS	Queries the RGB values of a single specified pixel value
QUERY COLORS	QUERY COLORS	Queries the RGB values of an array of pixels stored in the color data structures
QUERY EXTENSION	QUERY EXTENSION	Determines if the named extension is present and, if so, returns major opcode for the extension
QUERY POINTER	QUERY POINTER	Obtains the root window the pointer is currently on and the pointer coordinates relative to the root's origin
QUERY TEXT EXTENTS	QUERY TEXT EXTENTS	Queries the server for the bounding box of a 1-byte character string

(continued on next page)

Table B–1 (Cont.) Routines Requiring Protocol Requests

Xlib Function	Protocol Request	Description
QUERY TEXT EXTENTS 16	QUERY TEXT EXTENTS	Queries the server for the bounding box of a 2-byte character string in the specified font
QUERY TREE	QUERY TREE	Obtains a list of children, the parent, and number of children for a specified window
RAISE WINDOW	CONFIGURE WINDOW	Raises a window so that no sibling window obscures it
RECOLOR CURSOR	RECOLOR CURSOR	Changes the color of the specified cursor
RESIZE WINDOW	CONFIGURE WINDOW	Changes a window's size without changing the upper left coordinate
RESTACK WINDOWS	CONFIGURE WINDOW	Restacks a set of windows from top to bottom
ROTATE BUFFERS	ROTATE PROPERTIES	Rotates the cut buffers
ROTATE WINDOW PROPERTIES	ROTATE PROPERTIES	Rotates properties in the properties array
SELECT INPUT	CHANGE WINDOW ATTRIBUTES	Requests server to report events associated with the event masks passed to the event_mask argument
SEND EVENT	SEND EVENT	Sends an event to a specified window
SET ARC MODE	CHANGE GC	Sets the arc mode of the specified graphics context
SET BACKGROUND	CHANGE GC	Sets the background of the specified graphics context
SET CLIP MASK	CHANGE GC	Sets the clip_mask of the specified graphics context to the specified pixmap
SET CLIP ORIGIN	CHANGE GC	Sets the clip origin of the specified graphics context
SET CLIP RECTANGLES	SET CLIP RECTANGLES	Sets the clip_mask of the specified context to the specified list of rectangles
SET COMMAND	CHANGE PROPERTY	Sets the value of the command atom
SET DASHES	SET DASHES	Sets the dash_offset and dash_list for dashed line styles of the specified graphics context
SET FILL RULE	CHANGE GC	Sets the fill rule of the specified graphics context
SET FILL STYLE	CHANGE GC	Sets the fill style of the specified graphics context

(continued on next page)

Table B–1 (Cont.) Routines Requiring Protocol Requests

Xlib Function	Protocol Request	Description
SET FONT	CHANGE GC	Sets the current font of the specified graphics context
SET FONT PATH	SET FONT PATH	Sets the font search path
SET FOREGROUND	CHANGE GC	Sets the foreground of the specified graphics context
SET FUNCTION	CHANGE GC	Sets the display function in the specified graphics context
SET GRAPHICS EXPOSURES	CHANGE GC	Sets the graphics exposures flag of the specified graphics context
SET ICON SIZES	CHANGE PROPERTY	Sets the value of the icon size atom
SET LINE ATTRIBUTES	CHANGE GC	Sets the line drawing components of the specified graphics context
SET NORMAL HINTS	CHANGE PROPERTY	Sets the size hints for a window in its normal state
SET PLANE MASK	CHANGE GC	Sets the plane mask of the specified graphics context
SET SELECTION OWNER	SET SELECTION OWNER	Sets the selection owner
SET SIZE HINTS	CHANGE PROPERTY	Sets the value of any property of type WM_SIZE_HINTS
SET STANDARD PROPERTIES	CHANGE PROPERTY	Specifies a minimum set of properties describing a simple application
SET STATE	CHANGE GC	Sets the foreground, background, plane mask, and function components for the specified graphics context
SET STIPPLE	CHANGE GC	Sets the stipple of the specified graphics context
SET SUBWINDOW MODE	CHANGE GC	Sets the subwindow mode of the specified graphics context
SET TILE	CHANGE GC	Sets the fill tile of the specified graphics context
SET TS ORIGIN	CHANGE GC	Sets the tile or stipple origin of the specified graphics context
SET WM HINTS	CHANGE PROPERTY	Sets the value of the window manager hints atom
SET WINDOW BACKGROUND	CHANGE WINDOW ATTRIBUTES	Sets the background of a specified window to the specified pixel
SET WINDOW BACKGROUND PIXMAP	CHANGE WINDOW ATTRIBUTES	Sets the background of a specified window to the specified pixmap
SET WINDOW BORDER	CHANGE WINDOW ATTRIBUTES	Changes and repaints a window's border to the specified pixel

(continued on next page)

Table B–1 (Cont.) Routines Requiring Protocol Requests

Xlib Function	Protocol Request	Description
SET WINDOW BORDER PIXMAP	CHANGE WINDOW ATTRIBUTES	Changes and repaints a window's border tile
SET WINDOW BORDER WIDTH	CONFIGURE WINDOW	Changes the border width of a window
SET WINDOW COLORMAP	CHANGE WINDOW ATTRIBUTES	Sets the color map of a specified window
SET ZOOM HINTS	CHANGE PROPERTY	Sets the value of the zoom hints atom
STORE BUFFER	CHANGE PROPERTY	Stores data in specified cut buffer
STORE BYTES	CHANGE PROPERTY	Stores data in cut buffer zero
STORE COLOR	STORE COLORS	Stores an RGB value into a single color map cell
STORE COLORS	STORE COLORS	Stores RGB values into color map cells
STORE NAME	CHANGE PROPERTY	Assigns a name to a window
STORE NAMED COLOR	STORE NAMED COLOR	Sets the color of a pixel to the named color
SYNC	GET INPUT FOCUS	Flushes the output buffer and then waits until all requests have been processed
TRANSLATE COORDINATES	TRANSLATE COORDINATES	Performs a coordinate transformation from the coordinate space of one window to another window
UNDEFINE CURSOR	CHANGE WINDOW ATTRIBUTES	Removes the association of the cursor with the specified window
UNLOAD FONT	CLOSE FONT	Unloads the specified font that was loaded by LOAD FONT
UNMAP SUBWINDOWS	UNMAP SUBWINDOWS	Unmaps all subwindows for a specified window
UNMAP WINDOW	UNMAP WINDOW	Unmaps a window

C VMS DECwindows Named Colors

Table C–1 lists available VMS DECwindows named colors. The table provides the color name and the RGB values associated with that color. For a description of using named colors, see Section 5.3.1.

Table C–1 VMS DECwindows Named Colors

Named Color	RGB Values		
	Red	Green	Blue
Aquamarine	28672	56064	37632
MediumAquamarine	12800	52224	39168
Medium Aquamarine	12800	52224	39168
Black	0	0	0
Blue	0	0	65280
CadetBlue	24320	40704	40704
Cadet Blue	24320	40704	40704
CornflowerBlue	16896	16896	28416
Cornflower Blue	16896	16896	28416
DarkSlateBlue	27392	8960	36352
Dark Slate Blue	27392	8960	36352
LightBlue	48896	55296	55296
Light Blue	48896	55296	55296
LightSteelBlue	36608	36608	48128
Light Steel Blue	36608	36608	48128
MediumBlue	12800	12800	52224
Medium Blue	12800	12800	52224
MediumSlateBlue	32512	0	65280
Medium Slate Blue	32512	0	65280
MidnightBlue	12032	12032	20224
Midnight Blue	12032	12032	20224
NavyBlue	8960	8960	36352
Navy Blue	8960	8960	36352
Navy	8960	8960	36352
SkyBlue	12800	39168	52224
Sky Blue	12800	39168	52224
SlateBlue	0	32512	65280

(continued on next page)

Table C–1 (Cont.) VMS DECwindows Named Colors

Named Color	RGB Values		
	Red	Green	Blue
Slate Blue	0	32512	65280
SteelBlue	8960	27392	36352
Steel Blue	8960	27392	36352
Brown	42240	10752	10752
SandyBrown	62464	41984	24576
Coral	65280	32512	0
Cyan	0	65280	65280
Firebrick	36352	8960	8960
Gold	52224	32512	12800
Goldenrod	56064	56064	28672
MediumGoldenrod	59904	59904	44288
Medium Goldenrod	59904	59904	44288
Green	0	65280	0
DarkGreen	12032	20224	12032
Dark Green	12032	20224	12032
DarkOliveGreen	20224	20224	12032
Dark Olive Green	20224	20224	12032
ForestGreen	8960	36352	8960
Forest Green	8960	36352	8960
LimeGreen	12800	52224	12800
Lime Green	12800	52224	12800
MediumForestGreen	27392	36352	8960
Medium Forest Green	27392	36352	8960
MediumSeaGreen	16896	28416	16896
Medium Sea Green	16896	28416	16896
MediumSpringGreen	32512	65280	0
Medium Spring Green	32512	65280	0
PaleGreen	36608	48128	36608
Pale Green	36608	48128	36608
SeaGreen	8960	36352	27392
Sea Green	8960	36352	27392
SpringGreen	0	65280	32512
Spring Green	0	65280	32512
YellowGreen	39168	52224	12800
Yellow Green	39168	52224	12800

(continued on next page)

Table C–1 (Cont.) VMS DECwindows Named Colors

Named Color	RGB Values		
	Red	Green	Blue
DarkSlateGray	12032	20224	20224
Dark Slate Gray	12032	20224	20224
Dark Slate Grey	12032	20224	20224
DarkSlateGrey	12032	20224	20224
DimGray	21504	21504	21504
Dim Gray	21504	21504	21504
DimGrey	21504	21504	21504
Dim Grey	21504	21504	21504
LightGray	43008	43008	43008
Light Gray	43008	43008	43008
LightGrey	43008	43008	43008
Light Grey	43008	43008	43008
Khaki	40704	40704	24320
Magenta	65280	0	65280
Maroon	36352	8960	27392
Orange	52224	12800	12800
Orchid	56064	28672	56064
DarkOrchid	39168	12800	52224
Dark Orchid	39168	12800	52224
MediumOrchid	37632	28672	56064
Medium Orchid	37632	28672	56064
Pink	48128	36608	36608
Plum	59904	44288	59904
Red	65280	0	0
IndianRed	20224	12032	12032
Indian Red	20224	12032	12032
MediumVioletRed	56064	28672	37632
Medium Violet Red	56064	28672	37632
OrangeRed	65280	0	32512
Orange Red	65280	0	32512
VioletRed	52224	12800	39168
Violet Red	52224	12800	39168
Salmon	28416	16896	16896
Sienna	36352	27392	8960
Tan	56064	37632	28672

(continued on next page)

Table C–1 (Cont.) VMS DECwindows Named Colors

Named Color	RGB Values		
	Red	Green	Blue
Thistle	55296	48896	55296
Turquoise	44288	59904	59904
DarkTurquoise	28672	37632	56064
Dark Turquoise	28672	37632	56064
MediumTurquoise	28672	56064	56064
Medium Turquoise	28672	56064	56064
Violet	20224	12032	20224
BlueViolet	40704	24320	40704
Blue Violet	40704	24320	40704
Wheat	55296	55296	48896
White	65535	65535	65535
Yellow	65280	65280	0
GreenYellow	37632	56064	28672
Green Yellow	37632	56064	28672

D VMS DECwindows Fonts

Table D–1 lists VMS DECwindows 75 DPI fonts and their file names. Table D–2 lists VMS DECwindows 100 DPI fonts and their file names. For information about using fonts, see Chapter 8.

Table D–1 VMS DECwindows 75 DPI Fonts

File Name	Font Name
FIXED	FIXED (MIT) (now ISOLATIN1)
CURSOR	CURSOR (MIT)
DECW$CURSOR	DECW$CURSOR (VMS)
DECW$SESSION	DECW$SESSION (VMS)
VARIABLE	VARIABLE (MIT)
AVANT GARDE	
AVANTGARDE_BOOK8	-Adobe-ITC Avant Garde Gothic-Book-R-Normal–8-80-75-75-P-49-ISO8859-1
AVANTGARDE_BOOK10	-Adobe-ITC Avant Garde Gothic-Book-R-Normal–10-100-75-75-P-59-ISO8859-1
AVANTGARDE_BOOK12	-Adobe-ITC Avant Garde Gothic-Book-R-Normal–12-120-75-75-P-70-ISO8859-1
AVANTGARDE_BOOK14	-Adobe-ITC Avant Garde Gothic-Book-R-Normal–14-140-75-75-P-80-ISO8859-1
AVANTGARDE_BOOK18	-Adobe-ITC Avant Garde Gothic-Book-R-Normal–18-180-75-75-P-103-ISO8859-1
AVANTGARDE_BOOK24	-Adobe-ITC Avant Garde Gothic-Book-R-Normal–24-240-75-75-P-138-ISO8859-1
AVANTGARDE_BOOKOBLIQUE8	-Adobe-ITC Avant Garde Gothic-Book-O-Normal–8-80-75-75-P-49-ISO8859-1
AVANTGARDE_BOOKOBLIQUE10	-Adobe-ITC Avant Garde Gothic-Book-O-Normal–10-100-75-75-P-59-ISO8859-1
AVANTGARDE_BOOKOBLIQUE12	-Adobe-ITC Avant Garde Gothic-Book-O-Normal–12-120-75-75-P-69-ISO8859-1
AVANTGARDE_BOOKOBLIQUE14	-Adobe-ITC Avant Garde Gothic-Book-O-Normal–14-140-75-75-P-81-ISO8859-1
AVANTGARDE_BOOKOBLIQUE18	-Adobe-ITC Avant Garde Gothic-Book-O-Normal–18-180-75-75-P-103-ISO8859-1
AVANTGARDE_BOOKOBLIQUE24	-Adobe-ITC Avant Garde Gothic-Book-O-Normal–24-240-75-75-P-138-ISO8859-1
AVANTGARDE_DEMI8	-Adobe-ITC Avant Garde Gothic-Demi-R-Normal–8-80-75-75-P-51-ISO8859-1
AVANTGARDE_DEMI10	-Adobe-ITC Avant Garde Gothic-Demi-R-Normal–10-100-75-75-P-61-ISO8859-1
AVANTGARDE_DEMI12	-Adobe-ITC Avant Garde Gothic-Demi-R-Normal–12-120-75-75-P-70-ISO8859-1
AVANTGARDE_DEMI14	-Adobe-ITC Avant Garde Gothic-Demi-R-Normal–14-140-75-75-P-82-ISO8859-1
AVANTGARDE_DEMI18	-Adobe-ITC Avant Garde Gothic-Demi-R-Normal–18-180-75-75-P-105-ISO8859-1
AVANTGARDE_DEMI24	-Adobe-ITC Avant Garde Gothic-Demi-R-Normal–24-240-75-75-P-140-ISO8859-1
AVANTGARDE_DEMIOBLIQUE8	-Adobe-ITC Avant Garde Gothic-Demi-O-Normal–8-80-75-75-P-51-ISO8859-1
AVANTGARDE_DEMIOBLIQUE10	-Adobe-ITC Avant Garde Gothic-Demi-O-Normal–10-100-75-75-P-61-ISO8859-1

(continued on next page)

Table D–1 (Cont.) VMS DECwindows 75 DPI Fonts

File Name	Font Name
AVANT GARDE	
AVANTGARDE_DEMIOBLIQUE12	-Adobe-ITC Avant Garde Gothic-Demi-O-Normal–12-120-75-75-P-71-ISO8859-1
AVANTGARDE_DEMIOBLIQUE14	-Adobe-ITC Avant Garde Gothic-Demi-O-Normal–14-140-75-75-P-82-ISO8859-1
AVANTGARDE_DEMIOBLIQUE18	-Adobe-ITC Avant Garde Gothic-Demi-O-Normal–18-180-75-75-P-103-ISO8859-1
AVANTGARDE_DEMIOBLIQUE24	-Adobe-ITC Avant Garde Gothic-Demi-O-Normal–24-240-75-75-P-139-ISO8859-1
COURIER	
COURIER10	-Adobe-Courier-Medium-R-Normal–10-100-75-75-M-60-ISO8859-1
COURIER12	-Adobe-Courier-Medium-R-Normal–12-120-75-75-M-70-ISO8859-1
COURIER14	-Adobe-Courier-Medium-R-Normal–14-140-75-75-M-90-ISO8859-1
COURIER18	-Adobe-Courier-Medium-R-Normal–18-180-75-75-M-110-ISO8859-1
COURIER24	-Adobe-Courier-Medium-R-Normal–24-240-75-75-M-150-ISO8859-1
COURIER8	-Adobe-Courier-Medium-R-Normal–8-80-75-75-M-50-ISO8859-1
COURIER_BOLD10	-Adobe-Courier-Bold-R-Normal–10-100-75-75-M-60-ISO8859-1
COURIER_BOLD12	-Adobe-Courier-Bold-R-Normal–12-120-75-75-M-70-ISO8859-1
COURIER_BOLD14	-Adobe-Courier-Bold-R-Normal–14-140-75-75-M-90-ISO8859-1
COURIER_BOLD18	-Adobe-Courier-Bold-R-Normal–18-180-75-75-M-110-ISO8859-1
COURIER_BOLD24	-Adobe-Courier-Bold-R-Normal–24-240-75-75-M-150-ISO8859-1
COURIER_BOLD8	-Adobe-Courier-Bold-R-Normal–8-80-75-75-M-50-ISO8859-1
COURIER_BOLDOBLIQUE10	-Adobe-Courier-Bold-O-Normal–10-100-75-75-M-60-ISO8859-1
COURIER_BOLDOBLIQUE12	-Adobe-Courier-Bold-O-Normal–12-120-75-75-M-70-ISO8859-1
COURIER_BOLDOBLIQUE14	-Adobe-Courier-Bold-O-Normal–14-140-75-75-M-90-ISO8859-1
COURIER_BOLDOBLIQUE18	-Adobe-Courier-Bold-O-Normal–18-180-75-75-M-110-ISO8859-1
COURIER_BOLDOBLIQUE24	-Adobe-Courier-Bold-O-Normal–24-240-75-75-M-150-ISO8859-1
COURIER_BOLDOBLIQUE8	-Adobe-Courier-Bold-O-Normal–8-80-75-75-M-50-ISO8859-1
COURIER_OBLIQUE10	-Adobe-Courier-Medium-O-Normal–10-100-75-75-M-60-ISO8859-1
COURIER_OBLIQUE12	-Adobe-Courier-Medium-O-Normal–12-120-75-75-M-70-ISO8859-1
COURIER_OBLIQUE14	-Adobe-Courier-Medium-O-Normal–14-140-75-75-M-90-ISO8859-1
COURIER_OBLIQUE18	-Adobe-Courier-Medium-O-Normal–18-180-75-75-M-110-ISO8859-1
COURIER_OBLIQUE24	-Adobe-Courier-Medium-O-Normal–24-240-75-75-M-150-ISO8859-1
COURIER_OBLIQUE8	-Adobe-Courier-Medium-O-Normal–8-80-75-75-M-50-ISO8859-1

(continued on next page)

Table D–1 (Cont.) VMS DECwindows 75 DPI Fonts

File Name	Font Name
HELVETICA	
HELVETICA10	-ADOBE-Helvetica-Medium-R-Normal–10-100-75-75-P-56-ISO8859-1
HELVETICA12	-ADOBE-Helvetica-Medium-R-Normal–12-120-75-75-P-67-ISO8859-1
HELVETICA14	-ADOBE-Helvetica-Medium-R-Normal–14-140-75-75-P-77-ISO8859-1
HELVETICA18	-ADOBE-Helvetica-Medium-R-Normal–18-180-75-75-P-98-ISO8859-1
HELVETICA24	-ADOBE-Helvetica-Medium-R-Normal–24-240-75-75-P-130-ISO8859-1
HELVETICA8	-ADOBE-Helvetica-Medium-R-Normal–8-80-75-75-P-46-ISO8859-1
HELVETICA_BOLD10	-ADOBE-Helvetica-Bold-R-Normal–10-100-75-75-P-60-ISO8859-1
HELVETICA_BOLD12	-ADOBE-Helvetica-Bold-R-Normal–12-120-75-75-P-70-ISO8859-1
HELVETICA_BOLD14	-ADOBE-Helvetica-Bold-R-Normal–14-140-75-75-P-82-ISO8859-1
HELVETICA_BOLD18	-ADOBE-Helvetica-Bold-R-Normal–18-180-75-75-P-103-ISO8859-1
HELVETICA_BOLD24	-ADOBE-Helvetica-Bold-R-Normal–24-240-75-75-P-138-ISO8859-1
HELVETICA_BOLD8	-ADOBE-Helvetica-Bold-R-Normal–8-80-75-75-P-50-ISO8859-1
HELVETICA_BOLDOBLIQUE10	-ADOBE-Helvetica-Bold-O-Normal–10-100-75-75-P-60-ISO8859-1
HELVETICA_BOLDOBLIQUE12	-ADOBE-Helvetica-Bold-O-Normal–12-120-75-75-P-69-ISO8859-1
HELVETICA_BOLDOBLIQUE14	-ADOBE-Helvetica-Bold-O-Normal–14-140-75-75-P-82-ISO8859-1
HELVETICA_BOLDOBLIQUE18	-ADOBE-Helvetica-Bold-O-Normal–18-180-75-75-P-104-ISO8859-1
HELVETICA_BOLDOBLIQUE24	-ADOBE-Helvetica-Bold-O-Normal–24-240-75-75-P-138-ISO8859-1
HELVETICA_BOLDOBLIQUE8	-ADOBE-Helvetica-Bold-O-Normal–8-80-75-75-P-50-ISO8859-1
HELVETICA_OBLIQUE10	-ADOBE-Helvetica-Medium-O-Normal–10-100-75-75-P-57-ISO8859-1
HELVETICA_OBLIQUE12	-ADOBE-Helvetica-Medium-O-Normal–12-120-75-75-P-67-ISO8859-1
HELVETICA_OBLIQUE14	-ADOBE-Helvetica-Medium-O-Normal–14-140-75-75-P-78-ISO8859-1
HELVETICA_OBLIQUE18	-ADOBE-Helvetica-Medium-O-Normal–18-180-75-75-P-98-ISO8859-1
HELVETICA_OBLIQUE24	-ADOBE-Helvetica-Medium-O-Normal–24-240-75-75-P-130-ISO8859-1
HELVETICA_OBLIQUE8	-ADOBE-Helvetica-Medium-O-Normal–8-80-75-75-P-47-ISO8859-1
INTERIM	
INTERIM_DM_EXTENSION14	-ADOBE-Interim DM-Medium-I-Normal–14-140-75-75-P-140-DEC-DECMATH_EXTENSION
INTERIM_DM_ITALIC14	-ADOBE-Interim DM-Medium-I-Normal–14-140-75-75-P-140-DEC-DECMATH_ITALIC
INTERIM_DM_SYMBOL14	-ADOBE-Interim DM-Medium-I-Normal–14-140-75-75-P-140-DEC-DECMATH_SYMBOL
LUBALIN GRAPH	
LUBALINGRAPH_BOOK8	-Adobe-ITC Lubalin Graph-Book-R-Normal–8-80-75-75-P-50-ISO8859-1
LUBALINGRAPH_BOOK10	-Adobe-ITC Lubalin Graph-Book-R-Normal–10-100-75-75-P-60-ISO8859-1
LUBALINGRAPH_BOOK12	-Adobe-ITC Lubalin Graph-Book-R-Normal–12-120-75-75-P-70-ISO8859-1

(continued on next page)

Table D–1 (Cont.) VMS DECwindows 75 DPI Fonts

File Name	Font Name
LUBALIN GRAPH	
LUBALINGRAPH_BOOK14	-Adobe-ITC Lubalin Graph-Book-R-Normal–14-140-75-75-P-81-ISO8859-1
LUBALINGRAPH_BOOK18	-Adobe-ITC Lubalin Graph-Book-R-Normal–18-180-75-75-P-106-ISO8859-1
LUBALINGRAPH_BOOK24	-Adobe-ITC Lubalin Graph-Book-R-Normal–24-240-75-75-P-139-ISO8859-1
LUBALINGRAPH_BOOKOBLIQUE8	-Adobe-ITC Lubalin Graph-Book-O-Normal–8-80-75-75-P-50-ISO8859-1
LUBALINGRAPH_BOOKOBLIQUE10	-Adobe-ITC Lubalin Graph-Book-O-Normal–10-100-75-75-P-60-ISO8859-1
LUBALINGRAPH_BOOKOBLIQUE12	-Adobe-ITC Lubalin Graph-Book-O-Normal–12-120-75-75-P-70-ISO8859-1
LUBALINGRAPH_BOOKOBLIQUE14	-Adobe-ITC Lubalin Graph-Book-O-Normal–14-140-75-75-P-82-ISO8859-1
LUBALINGRAPH_BOOKOBLIQUE18	-Adobe-ITC Lubalin Graph-Book-O-Normal–18-180-75-75-P-105-ISO8859-1
LUBALINGRAPH_BOOKOBLIQUE24	-Adobe-ITC Lubalin Graph-Book-O-Normal–24-240-75-75-P-140-ISO8859-1
LUBALINGRAPH_DEMI8	-Adobe-ITC Lubalin Graph-Demi-R-Normal–8-80-75-75-P-51-ISO8859-1
LUBALINGRAPH_DEMI10	-Adobe-ITC Lubalin Graph-Demi-R-Normal–10-100-75-75-P-61-ISO8859-1
LUBALINGRAPH_DEMI12	-Adobe-ITC Lubalin Graph-Demi-R-Normal–12-120-75-75-P-73-ISO8859-1
LUBALINGRAPH_DEMI14	-Adobe-ITC Lubalin Graph-Demi-R-Normal–14-140-75-75-P-85-ISO8859-1
LUBALINGRAPH_DEMI18	-Adobe-ITC Lubalin Graph-Demi-R-Normal–18-180-75-75-P-109-ISO8859-1
LUBALINGRAPH_DEMI24	-Adobe-ITC Lubalin Graph-Demi-R-Normal–24-240-75-75-P-144-ISO8859-1
LUBALINGRAPH_DEMIOBLIQUE8	-Adobe-ITC Lubalin Graph-Demi-O-Normal–8-80-75-75-P-52-ISO8859-1
LUBALINGRAPH_DEMIOBLIQUE10	-Adobe-ITC Lubalin Graph-Demi-O-Normal–10-100-75-75-P-62-ISO8859-1
LUBALINGRAPH_DEMIOBLIQUE12	-Adobe-ITC Lubalin Graph-Demi-O-Normal–12-120-75-75-P-74-ISO8859-1
LUBALINGRAPH_DEMIOBLIQUE14	-Adobe-ITC Lubalin Graph-Demi-O-Normal–14-140-75-75-P-85-ISO8859-1
LUBALINGRAPH_DEMIOBLIQUE18	-Adobe-ITC Lubalin Graph-Demi-O-Normal–18-180-75-75-P-109-ISO8859-1
LUBALINGRAPH_DEMIOBLIQUE24	-Adobe-ITC Lubalin Graph-Demi-O-Normal–24-240-75-75-P-144-ISO8859-1
MENU	
MENU10	-Bigelow & Holmes-Menu-Medium-R-Normal–10-100-75-75-P-56-ISO8859-1
MENU12	-Bigelow & Holmes-Menu-Medium-R-Normal–12-120-75-75-P-70-ISO8859-1
NEW CENTURY SCHOOLBOOK	
NEWCENTURYSCHLBK_BOLD10	-Adobe-New Century Schoolbook-Bold-R-Normal–10-100-75-75-P-66-ISO8859-1
NEWCENTURYSCHLBK_BOLD12	-Adobe-New Century Schoolbook-Bold-R-Normal–12-120-75-75-P-77-ISO8859-1
NEWCENTURYSCHLBK_BOLD14	-Adobe-New Century Schoolbook-Bold-R-Normal–14-140-75-75-P-87-ISO8859-1
NEWCENTURYSCHLBK_BOLD18	-Adobe-New Century Schoolbook-Bold-R-Normal–18-180-75-75-P-113-ISO8859-1
NEWCENTURYSCHLBK_BOLD24	-Adobe-New Century Schoolbook-Bold-R-Normal–24-240-75-75-P-149-ISO8859-1
NEWCENTURYSCHLBK_BOLD8	-Adobe-New Century Schoolbook-Bold-R-Normal–8-80-75-75-P-56-ISO8859-1
NEWCENTURYSCHLBK_BOLDITALIC10	-Adobe-New Century Schoolbook-Bold-I-Normal–10-100-75-75-P-66-ISO8859-1

(continued on next page)

Table D–1 (Cont.) VMS DECwindows 75 DPI Fonts

File Name	Font Name
NEW CENTURY SCHOOLBOOK	
NEWCENTURYSCHLBK_BOLDITALIC12	-Adobe-New Century Schoolbook-Bold-I-Normal–12-120-75-75-P-76-ISO8859-1
NEWCENTURYSCHLBK_BOLDITALIC14	-Adobe-New Century Schoolbook-Bold-I-Normal–14-140-75-75-P-88-ISO8859-1
NEWCENTURYSCHLBK_BOLDITALIC18	-Adobe-New Century Schoolbook-Bold-I-Normal–18-180-75-75-P-111-ISO8859-1
NEWCENTURYSCHLBK_BOLDITALIC24	-Adobe-New Century Schoolbook-Bold-I-Normal–24-240-75-75-P-148-ISO8859-1
NEWCENTURYSCHLBK_BOLDITALIC8	-Adobe-New Century Schoolbook-Bold-I-Normal–8-80-75-75-P-56-ISO8859-1
NEWCENTURYSCHLBK_ITALIC10	-Adobe-New Century Schoolbook-Medium-I-Normal–10-100-75-75-P-60-ISO8859-1
NEWCENTURYSCHLBK_ITALIC12	-Adobe-New Century Schoolbook-Medium-I-Normal–12-120-75-75-P-70-ISO8859-1
NEWCENTURYSCHLBK_ITALIC14	-Adobe-New Century Schoolbook-Medium-I-Normal–14-140-75-75-P-81-ISO8859-1
NEWCENTURYSCHLBK_ITALIC18	-Adobe-New Century Schoolbook-Medium-I-Normal–18-180-75-75-P-104-ISO8859-1
NEWCENTURYSCHLBK_ITALIC24	-Adobe-New Century Schoolbook-Medium-I-Normal–24-240-75-75-P-136-ISO8859-1
NEWCENTURYSCHLBK_ITALIC8	-Adobe-New Century Schoolbook-Medium-I-Normal–8-80-75-75-P-50-ISO8859-1
NEWCENTURYSCHLBK_ROMAN10	-Adobe-New Century Schoolbook-Medium-R-Normal–10-100-75-75-P-60-ISO8859-1
NEWCENTURYSCHLBK_ROMAN12	-Adobe-New Century Schoolbook-Medium-R-Normal–12-120-75-75-P-70-ISO8859-1
NEWCENTURYSCHLBK_ROMAN14	-Adobe-New Century Schoolbook-Medium-R-Normal–14-140-75-75-P-82-ISO8859-1
NEWCENTURYSCHLBK_ROMAN18	-Adobe-New Century Schoolbook-Medium-R-Normal–18-180-75-75-P-103-ISO8859-1
NEWCENTURYSCHLBK_ROMAN24	-Adobe-New Century Schoolbook-Medium-R-Normal–24-240-75-75-P-137-ISO8859-1
NEWCENTURYSCHLBK_ROMAN8	-Adobe-New Century Schoolbook-Medium-R-Normal–8-80-75-75-P-50-ISO8859-1
SOUVENIR	
SOUVENIR_DEMI10	-Adobe-ITC Souvenir-Demi-R-Normal–10-100-75-75-P-62-ISO8859-1
SOUVENIR_DEMI12	-Adobe-ITC Souvenir-Demi-R-Normal–12-120-75-75-P-75-ISO8859-1
SOUVENIR_DEMI14	-Adobe-ITC Souvenir-Demi-R-Normal–14-140-75-75-P-90-ISO8859-1
SOUVENIR_DEMI18	-Adobe-ITC Souvenir-Demi-R-Normal–18-180-75-75-P-112-ISO8859-1
SOUVENIR_DEMI24	-Adobe-ITC Souvenir-Demi-R-Normal–24-240-75-75-P-149-ISO8859-1
SOUVENIR_DEMI8	-Adobe-ITC Souvenir-Demi-R-Normal–8-80-75-75-P-52-ISO8859-1
SOUVENIR_DEMIITALIC10	-Adobe-ITC Souvenir-Demi-I-Normal–10-100-75-75-P-67-ISO8859-1
SOUVENIR_DEMIITALIC12	-Adobe-ITC Souvenir-Demi-I-Normal–12-120-75-75-P-78-ISO8859-1
SOUVENIR_DEMIITALIC14	-Adobe-ITC Souvenir-Demi-I-Normal–14-140-75-75-P-92-ISO8859-1
SOUVENIR_DEMIITALIC18	-Adobe-ITC Souvenir-Demi-I-Normal–18-180-75-75-P-115-ISO8859-1
SOUVENIR_DEMIITALIC24	-Adobe-ITC Souvenir-Demi-I-Normal–24-240-75-75-P-154-ISO8859-1
SOUVENIR_DEMIITALIC8	-Adobe-ITC Souvenir-Demi-I-Normal–8-80-75-75-P-57-ISO8859-1
SOUVENIR_LIGHT10	-Adobe-ITC Souvenir-Light-R-Normal–10-100-75-75-P-56-ISO8859-1
SOUVENIR_LIGHT12	-Adobe-ITC Souvenir-Light-R-Normal–12-120-75-75-P-68-ISO8859-1
SOUVENIR_LIGHT14	-Adobe-ITC Souvenir-Light-R-Normal–14-140-75-75-P-79-ISO8859-1
SOUVENIR_LIGHT18	-Adobe-ITC Souvenir-Light-R-Normal–18-180-75-75-P-102-ISO8859-1

(continued on next page)

Table D–1 (Cont.) VMS DECwindows 75 DPI Fonts

File Name	Font Name
SOUVENIR	
SOUVENIR_LIGHT24	-Adobe-ITC Souvenir-Light-R-Normal–24-240-75-75-P-135-ISO8859-1
SOUVENIR_LIGHT8	-Adobe-ITC Souvenir-Light-R-Normal–8-80-75-75-P-46-ISO8859-1
SOUVENIR_LIGHTITALIC10	-Adobe-ITC Souvenir-Light-I-Normal–10-100-75-75-P-59-ISO8859-1
SOUVENIR_LIGHTITALIC12	-Adobe-ITC Souvenir-Light-I-Normal–12-120-75-75-P-69-ISO8859-1
SOUVENIR_LIGHTITALIC14	-Adobe-ITC Souvenir-Light-I-Normal–14-140-75-75-P-82-ISO8859-1
SOUVENIR_LIGHTITALIC18	-Adobe-ITC Souvenir-Light-I-Normal–18-180-75-75-P-104-ISO8859-1
SOUVENIR_LIGHTITALIC24	-Adobe-ITC Souvenir-Light-I-Normal–24-240-75-75-P-139-ISO8859-1
SOUVENIR_LIGHTITALIC8	-Adobe-ITC Souvenir-Light-I-Normal–8-80-75-75-P-49-ISO8859-1
SYMBOL	
SYMBOL10	-Adobe-Symbol-Medium-R-Normal–10-100-75-75-P-61-ADOBE-FONTSPECIFIC
SYMBOL12	-Adobe-Symbol-Medium-R-Normal–12-120-75-75-P-74-ADOBE-FONTSPECIFIC
SYMBOL14	-ADOBE-Symbol-Medium-R-Normal–14-140-75-75-P-85-ADOBE-FONTSPECIFIC
SYMBOL18	-Adobe-Symbol-Medium-R-Normal–18-180-75-75-P-107-ADOBE-FONTSPECIFIC
SYMBOL24	-Adobe-Symbol-Medium-R-Normal–24-240-75-75-P-142-ADOBE-FONTSPECIFIC
SYMBOL8	-Adobe-Symbol-Medium-R-Normal–8-80-75-75-P-51-ADOBE-FONTSPECIFIC
TERMINAL	
TERMINAL14	-DEC-Terminal-Medium-R-Normal–14-140-75-75-C-8-ISO8859-1
TERMINAL18	-Bitstream-Terminal-Medium-R-Normal–18-180-75-75-C-11-ISO8859-1
TERMINAL28	-DEC-Terminal-Medium-R-Normal–28-280-75-75-C-16-ISO8859-1
TERMINAL36	-Bitstream-Terminal-Medium-R-Normal–36-360-75-75-C-22-ISO8859-1
TERMINAL_BOLD14	-DEC-Terminal-Bold-R-Normal–14-140-75-75-C-8-ISO8859-1
TERMINAL_BOLD18	-Bitstream-Terminal-Bold-R-Normal–18-180-75-75-C-11-ISO8859-1
TERMINAL_BOLD28	-DEC-Terminal-Bold-R-Normal–28-280-75-75-C-16-ISO8859-1
TERMINAL_BOLD36	-Bitstream-Terminal-Bold-R-Normal–36-360-75-75-C-22-ISO8859-1
TERMINAL_BOLD_DBLWIDE14	-DEC-Terminal-Bold-R-Double Wide–14-140-75-75-C-16-ISO8859-1
TERMINAL_BOLD_DBLWIDE18	-Bitstream-Terminal-Bold-R-Double Wide–18-180-75-75-C-22-ISO8859-1
TERMINAL_BOLD_DBLWIDE_DECTECH14	-DEC-Terminal-Bold-R-Double Wide–14-140-75-75-C-16-DEC-DECtech
TERMINAL_BOLD_DBLWIDE_DECTECH18	-Bitstream-Terminal-Bold-R-Double Wide–18-180-75-75-C-22-DEC-DECtech
TERMINAL_BOLD_DECTECH14	-DEC-Terminal-Bold-R-Normal–14-140-75-75-C-8-DEC-DECtech
TERMINAL_BOLD_DECTECH18	-Bitstream-Terminal-Bold-R-Normal–18-180-75-75-C-11-DEC-DECtech
TERMINAL_BOLD_DECTECH28	-DEC-Terminal-Bold-R-Normal–28-280-75-75-C-16-DEC-DECtech
TERMINAL_BOLD_DECTECH36	-Bitstream-Terminal-Bold-R-Normal–36-360-75-75-C-22-DEC-DECtech

(continued on next page)

Table D–1 (Cont.) VMS DECwindows 75 DPI Fonts

File Name	Font Name
TERMINAL	
TERMINAL_BOLD_NARROW14	-DEC-Terminal-Bold-R-Narrow–14-140-75-75-C-6-ISO8859-1
TERMINAL_BOLD_NARROW18	-Bitstream-Terminal-Bold-R-Narrow–18-180-75-75-C-7-ISO8859-1
TERMINAL_BOLD_NARROW28	-DEC-Terminal-Bold-R-Narrow–28-280-75-75-C-12-ISO8859-1
TERMINAL_BOLD_NARROW36	-Bitstream-Terminal-Bold-R-Narrow–36-360-75-75-C-14-ISO8859-1
TERMINAL_BOLD_NARROW_DECTECH14	-DEC-Terminal-Bold-R-Narrow–14-140-75-75-C-6-DEC-DECtech
TERMINAL_BOLD_NARROW_DECTECH18	-Bitstream-Terminal-Bold-R-Narrow–18-180-75-75-C-7-DEC-DECtech
TERMINAL_BOLD_NARROW_DECTECH28	-DEC-Terminal-Bold-R-Narrow–28-280-75-75-C-12-DEC-DECtech
TERMINAL_BOLD_NARROW_DECTECH36	-Bitstream-Terminal-Bold-R-Narrow–36-360-75-75-C-14-DEC-DECtech
TERMINAL_BOLD_WIDE14	-DEC-Terminal-Bold-R-Wide–14-140-75-75-C-12-ISO8859-1
TERMINAL_BOLD_WIDE18	-Bitstream-Terminal-Bold-R-Narrow–18-180-75-75-C-14-ISO8859-1
TERMINAL_BOLD_WIDE_DECTECH14	-DEC-Terminal-Bold-R-Wide–14-140-75-75-C-12-DEC-DECtech
TERMINAL_BOLD_WIDE_DECTECH18	-Bitstream-Terminal-Bold-R-Narrow–18-180-75-75-C-14-DEC-DECtech
TERMINAL_DBLWIDE14	-DEC-Terminal-Medium-R-Double Wide–14-140-75-75-C-16-ISO8859-1
TERMINAL_DBLWIDE18	-Bitstream-Terminal-Medium-R-Double Wide–18-180-75-75-C-22-ISO8859-1
TERMINAL_DBLWIDE_DECTECH14	-DEC-Terminal-Medium-R-Double Wide–14-140-75-75-C-16-DEC-DECtech
TERMINAL_DBLWIDE_DECTECH18	-Bitstream-Terminal-Medium-R-Double Wide–18-180-75-75-C-22-DEC-DECtech
TERMINAL_DECTECH14	-DEC-Terminal-Medium-R-Normal–14-140-75-75-C-8-DEC-DECtech
TERMINAL_DECTECH18	-Bitstream-Terminal-Medium-R-Normal–18-180-75-75-C-11-DEC-DECtech
TERMINAL_DECTECH28	-DEC-Terminal-Medium-R-Normal–28-280-75-75-C-16-DEC-DECtech
TERMINAL_DECTECH36	-Bitstream-Terminal-Medium-R-Normal–36-360-75-75-C-22-DEC-DECtech
TERMINAL_NARROW14	-DEC-Terminal-Medium-R-Narrow–14-140-75-75-C-6-ISO8859-1
TERMINAL_NARROW18	-Bitstream-Terminal-Medium-R-Narrow–18-180-75-75-C-7-ISO8859-1
TERMINAL_NARROW28	-DEC-Terminal-Medium-R-Narrow–28-280-75-75-C-12-ISO8859-1
TERMINAL_NARROW36	-Bitstream-Terminal-Medium-R-Narrow–36-360-75-75-C-14-ISO8859-1
TERMINAL_NARROW_DECTECH14	-DEC-Terminal-Medium-R-Narrow–14-140-75-75-C-6-DEC-DECtech
TERMINAL_NARROW_DECTECH18	-Bitstream-Terminal-Medium-R-Narrow–18-180-75-75-C-7-DEC-DECtech
TERMINAL_NARROW_DECTECH28	-DEC-Terminal-Medium-R-Narrow–28-280-75-75-C-12-DEC-DECtech
TERMINAL_NARROW_DECTECH36	-Bitstream-Terminal-Medium-R-Narrow–36-360-75-75-C-14-DEC-DECtech
TERMINAL_WIDE14	-DEC-Terminal-Medium-R-Wide–14-140-75-75-C-12-ISO8859-1
TERMINAL_WIDE18	-Bitstream-Terminal-Medium-R-Wide–18-180-75-75-C-14-ISO8859-1
TERMINAL_WIDE_DECTECH14	-DEC-Terminal-Medium-R-Wide–14-140-75-75-C-12-DEC-DECtech
TERMINAL_WIDE_DECTECH18	-Bitstream-Terminal-Medium-R-Wide–18-180-75-75-C-14-DEC-DECtech

(continued on next page)

Table D–1 (Cont.) VMS DECwindows 75 DPI Fonts

File Name	Font Name
TIMES	
TIMES_BOLD10	-ADOBE-Times-Bold-R-Normal–10-100-75-75-P-57-ISO8859-1
TIMES_BOLD12	-ADOBE-Times-Bold-R-Normal–12-120-75-75-P-67-ISO8859-1
TIMES_BOLD14	-ADOBE-Times-Bold-R-Normal–14-140-75-75-P-77-ISO8859-1
TIMES_BOLD18	-ADOBE-Times-Bold-R-Normal–18-180-75-75-P-99-ISO8859-1
TIMES_BOLD24	-ADOBE-Times-Bold-R-Normal–24-240-75-75-P-132-ISO8859-1
TIMES_BOLD8	-ADOBE-Times-Bold-R-Normal–8-80-75-75-P-47-ISO8859-1
TIMES_BOLDITALIC10	-ADOBE-Times-Bold-I-Normal–10-100-75-75-P-57-ISO8859-1
TIMES_BOLDITALIC12	-ADOBE-Times-Bold-I-Normal–12-120-75-75-P-68-ISO8859-1
TIMES_BOLDITALIC14	-ADOBE-Times-Bold-I-Normal–14-140-75-75-P-77-ISO8859-1
TIMES_BOLDITALIC18	-ADOBE-Times-Bold-I-Normal–18-180-75-75-P-98-ISO8859-1
TIMES_BOLDITALIC24	-ADOBE-Times-Bold-I-Normal–24-240-75-75-P-128-ISO8859-1
TIMES_BOLDITALIC8	-ADOBE-Times-Bold-I-Normal–8-80-75-75-P-47-ISO8859-1
TIMES_ITALIC10	-ADOBE-Times-Medium-I-Normal–10-100-75-75-P-52-ISO8859-1
TIMES_ITALIC12	-ADOBE-Times-Medium-I-Normal–12-120-75-75-P-63-ISO8859-1
TIMES_ITALIC14	-ADOBE-Times-Medium-I-Normal–14-140-75-75-P-73-ISO8859-1
TIMES_ITALIC18	-ADOBE-Times-Medium-I-Normal–18-180-75-75-P-94-ISO8859-1
TIMES_ITALIC24	-ADOBE-Times-Medium-I-Normal–24-240-75-75-P-125-ISO8859-1
TIMES_ITALIC8	-ADOBE-Times-Medium-I-Normal–8-80-75-75-P-42-ISO8859-1
TIMES_ROMAN10	-ADOBE-Times-Medium-R-Normal–10-100-75-75-P-54-ISO8859-1
TIMES_ROMAN12	-ADOBE-Times-Medium-R-Normal–12-120-75-75-P-64-ISO8859-1
TIMES_ROMAN14	-ADOBE-Times-Medium-R-Normal–14-140-75-75-P-74-ISO8859-1
TIMES_ROMAN18	-ADOBE-Times-Medium-R-Normal–18-180-75-75-P-94-ISO8859-1
TIMES_ROMAN24	-ADOBE-Times-Medium-R-Normal–24-240-75-75-P-124-ISO8859-1
TIMES_ROMAN8	-ADOBE-Times-Medium-R-Normal–8-80-75-75-P-44-ISO8859-1

Table D–2 VMS DECwindows 100 DPI Fonts

File Name	Font Name
FIXED_100DPI	FIXED (MIT)
CURSOR_100DPI	CURSOR (MIT)
DECW$CURSOR_100DPI	W$CURSOR (VMS)
VARIABLE_100DPI	VARIABLE (MIT)

(continued on next page)

Table D–2 (Cont.) VMS DECwindows 100 DPI Fonts

File Name	Font Name
AVANT GARDE	
AVANTGARDE_BOOK8_100DPI	-Adobe-ITC Adobe-ITC Avant Garde Gothic-Book-R-Normal–11-80-100-100-P-59-ISO8859-1
AVANTGARDE_BOOK10_100DPI	-Adobe-ITC Avant Garde Gothic-Book-R-Normal–14-100-100-100-P-80-ISO8859-1
AVANTGARDE_BOOK12_100DPI	-Adobe-ITC Avant Garde Gothic-Book-R-Normal–17-120-100-100-P-93-ISO8859-1
AVANTGARDE_BOOK14_100DPI	-Adobe-ITC Avant Garde Gothic-Book-R-Normal–20-140-100-100-P-104-ISO8859-1
AVANTGARDE_BOOK18_100DPI	-Adobe-ITC Avant Garde Gothic-Book-R-Normal–25-180-100-100-P-138-ISO8859-1
AVANTGARDE_BOOK24_100DPI	-Adobe-ITC Avant Garde Gothic-Book-R-Normal–34-240-100-100-P-183-ISO8859-1
AVANTGARDE_BOOKOBLIQUE8_100DPI	-Avant Garde Gothic-Book-O-Normal–10-80-100-100-P-59-ISO8859-1
AVANTGARDE_BOOKOBLIQUE10_100DPI	-Adobe-ITC Avant Garde Gothic-Book-O-Normal–14-100-100-100-P-81-ISO8859-1
AVANTGARDE_BOOKOBLIQUE12_100DPI	-Adobe-ITC Avant Garde Gothic-Book-O-Normal–17-120-100-100-P-92-ISO8859-1
AVANTGARDE_BOOKOBLIQUE14_100DPI	-Adobe-ITC Avant Garde Gothic-Book-O-Normal–20-140-100-100-P-103-ISO8859-1
AVANTGARDE_BOOKOBLIQUE18_100DPI	-Adobe-ITC Avant Garde Gothic-Book-O-Normal–25-180-100-100-P-138-ISO8859-1
AVANTGARDE_BOOKOBLIQUE24_100DPI	-Adobe-ITC Avant Garde Gothic-Book-O-Normal–34-240-100-100-P-184-ISO8859-1
AVANTGARDE_DEMI8_100DPI	-Adobe-ITC Avant Garde Gothic-Demi-R-Normal–11-80-100-100-P-61-ISO8859-1
AVANTGARDE_DEMI10_100DPI	-Adobe-ITC Avant Garde Gothic-Demi-R-Normal–14-100-100-100-P-82-ISO8859-1
AVANTGARDE_DEMI12_100DPI	-Adobe-ITC Avant Garde Gothic-Demi-R-Normal–17-120-100-100-P-93-ISO8859-1
AVANTGARDE_DEMI14_100DPI	-Adobe-ITC Avant Garde Gothic-Demi-R-Normal–20-140-100-100-P-105-ISO8859-1
AVANTGARDE_DEMI18_100DPI	-Adobe-ITC Avant Garde Gothic-Demi-R-Normal–25-180-100-100-P-140-ISO8859-1
AVANTGARDE_DEMI24_100DPI	-Adobe-ITC Avant Garde Gothic-Demi-R-Normal–34-240-100-100-P-182-ISO8859-1
AVANTGARDE_DEMIOBLIQUE8_100DPI	-Adobe-ITC Avant Garde Gothic-Demi-O-Normal–11-80-100-100-P-61-ISO8859-1
AVANTGARDE_DEMIOBLIQUE10_100DPI	-Adobe-ITC Avant Garde Gothic-Demi-O-Normal–14-100-100-100-P-82-ISO8859-1
AVANTGARDE_DEMIOBLIQUE12_100DPI	-Adobe-ITC Avant Garde Gothic-Demi-O-Normal–17-120-100-100-P-93-ISO8859-1
AVANTGARDE_DEMIOBLIQUE14_100DPI	-Adobe-ITC Avant Garde Gothic-Demi-O-Normal–20-140-100-100-P-103-ISO8859-1
AVANTGARDE_DEMIOBLIQUE18_100DPI	-Adobe-ITC Avant Garde Gothic-Demi-O-Normal–25-180-100-100-P-139-ISO8859-1
AVANTGARDE_DEMIOBLIQUE24_100DPI	-Adobe-ITC Avant Garde Gothic-Demi-O-Normal–34-240-100-100-P-183-ISO8859-1
COURIER	
COURIER8_100DPI	-Adobe-Courier-Medium-R-Normal–11-80-100-100-M-60-ISO8859-1
COURIER10_100DPI	-Adobe-Courier-Medium-R-Normal–14-100-100-100-M-90-ISO8859-1
COURIER12_100DPI	-Adobe-Courier-Medium-R-Normal–17-120-100-100-M-100-ISO8859-1
COURIER14_100DPI	-Adobe-Courier-Medium-R-Normal–20-140-100-100-M-110-ISO8859-1
COURIER18_100DPI	-Adobe-Courier-Medium-R-Normal–25-180-100-100-M-150-ISO8859-1
COURIER24_100DPI	-Adobe-Courier-Medium-R-Normal–34-240-100-100-M-200-ISO8859-1
COURIER_BOLD8_100DPI	-Adobe-Courier-Bold-R-Normal–11-80-100-100-M-60-ISO8859-1
COURIER_BOLD10_100DPI	-Adobe-Courier-Bold-R-Normal–14-100-100-100-M-90-ISO8859-1
COURIER_BOLD12_100DPI	-Adobe-Courier-Bold-R-Normal–17-120-100-100-M-100-ISO8859-1

(continued on next page)

Table D–2 (Cont.) VMS DECwindows 100 DPI Fonts

File Name	Font Name
COURIER	
COURIER_BOLD14_100DPI	-Adobe-Courier-Bold-R-Normal–20-140-100-100-M-110-ISO8859-1
COURIER_BOLD18_100DPI	-Adobe-Courier-Bold-R-Normal–25-180-100-100-M-150-ISO8859-1
COURIER_BOLD24_100DPI	-Adobe-Courier-Bold-R-Normal–34-240-100-100-M-200-ISO8859-1
COURIER_BOLDOBLIQUE8_100DPI	-Adobe-Courier-Bold-O-Normal–11-80-100-100-M-60-ISO8859-1
COURIER_BOLDOBLIQUE10_100DPI	-Adobe-Courier-Bold-O-Normal–14-100-100-100-M-90-ISO8859-1
COURIER_BOLDOBLIQUE12_100DPI	-Adobe-Courier-Bold-O-Normal–17-120-100-100-M-100-ISO8859-1
COURIER_BOLDOBLIQUE14_100DPI	-Adobe-Courier-Bold-O-Normal–20-140-100-100-M-110-ISO8859-1
COURIER_BOLDOBLIQUE18_100DPI	-Adobe-Courier-Bold-O-Normal–25-180-100-100-M-150-ISO8859-1
COURIER_BOLDOBLIQUE24_100DPI	-Adobe-Courier-Bold-O-Normal–34-240-100-100-M-200-ISO8859-1
COURIER_OBLIQUE8_100DPI	-Adobe-Courier-Medium-O-Normal–11-80-100-100-M-60-ISO8859-1
COURIER_OBLIQUE10_100DPI	-Adobe-Courier-Medium-O-Normal–14-100-100-100-M-90-ISO8859-1
COURIER_OBLIQUE12_100DPI	-Adobe-Courier-Medium-O-Normal–17-120-100-100-M-100-ISO8859-1
COURIER_OBLIQUE14_100DPI	-Adobe-Courier-Medium-O-Normal–20-140-100-100-M-110-ISO8859-1
COURIER_OBLIQUE18_100DPI	-Adobe-Courier-Medium-O-Normal–25-180-100-100-M-150-ISO8859-1
COURIER_OBLIQUE24_100DPI	-Adobe-Courier-Medium-O-Normal–34-240-100-100-M-200-ISO8859-1
HELVETICA	
HELVETICA10_100DPI	-Adobe-Helvetica-Medium-R-Normal–14-100-100-100-P-76-ISO8859-1
HELVETICA12_100DPI	-Adobe-Helvetica-Medium-R-Normal–17-120-100-100-P-88-ISO8859-1
HELVETICA14_100DPI	-Adobe-Helvetica-Medium-R-Normal–20-140-100-100-P-100-ISO8859-1
HELVETICA18_100DPI	-Adobe-Helvetica-Medium-R-Normal–25-180-100-100-P-130-ISO8859-1
HELVETICA24_100DPI	-Adobe-Helvetica-Medium-R-Normal–34-240-100-100-P-176-ISO8859-1
HELVETICA8_100DPI	-Adobe-Helvetica-Medium-R-Normal–11-80-100-100-P-56-ISO8859-1
HELVETICA_BOLD10_100DPI	-Adobe-Helvetica-Bold-R-Normal–14-100-100-100-P-82-ISO8859-1
HELVETICA_BOLD12_100DPI	-Adobe-Helvetica-Bold-R-Normal–17-120-100-100-P-92-ISO8859-1
HELVETICA_BOLD14_100DPI	-Adobe-Helvetica-Bold-R-Normal–20-140-100-100-P-105-ISO8859-1
HELVETICA_BOLD18_100DPI	-Adobe-Helvetica-Bold-R-Normal–25-180-100-100-P-138-ISO8859-1
HELVETICA_BOLD24_100DPI	-Adobe-Helvetica-Bold-R-Normal–34-240-100-100-P-182-ISO8859-1
HELVETICA_BOLD8_100DPI	-Adobe-Helvetica-Bold-R-Normal–11-80-100-100-P-60-ISO8859-1
HELVETICA_BOLDOBLIQUE10_100DPI	-Adobe-Helvetica-Bold-O-Normal–14-100-100-100-P-82-ISO8859-1
HELVETICA_BOLDOBLIQUE12_100DPI	-Adobe-Helvetica-Bold-O-Normal–17-120-100-100-P-92-ISO8859-1
HELVETICA_BOLDOBLIQUE14_100DPI	-Adobe-Helvetica-Bold-O-Normal–20-140-100-100-P-103-ISO8859-1
HELVETICA_BOLDOBLIQUE18_100DPI	-Adobe-Helvetica-Bold-O-Normal–25-180-100-100-P-138-ISO8859-1
HELVETICA_BOLDOBLIQUE24_100DPI	-Adobe-Helvetica-Bold-O-Normal–34-240-100-100-P-182-ISO8859-1
HELVETICA_BOLDOBLIQUE8_100DPI	-Adobe-Helvetica-Bold-O-Normal–11-80-100-100-P-60-ISO8859-1

(continued on next page)

Table D–2 (Cont.) VMS DECwindows 100 DPI Fonts

File Name	Font Name
HELVETICA	
HELVETICA_OBLIQUE10_100DPI	-Adobe-Helvetica-Medium-O-Normal–14-100-100-100-P-78-ISO8859-1
HELVETICA_OBLIQUE12_100DPI	-Adobe-Helvetica-Medium-O-Normal–17-120-100-100-P-88-ISO8859-1
HELVETICA_OBLIQUE14_100DPI	-Adobe-Helvetica-Medium-O-Normal–20-140-100-100-P-98-ISO8859-1
HELVETICA_OBLIQUE18_100DPI	-Adobe-Helvetica-Medium-O-Normal–25-180-100-100-P-130-ISO8859-1
HELVETICA_OBLIQUE24_100DPI	-Adobe-Helvetica-Medium-O-Normal–34-240-100-100-P-176-ISO8859-1
HELVETICA_OBLIQUE8_100DPI	-Adobe-Helvetica-Medium-O-Normal–11-80-100-100-P-57-ISO8859-1
INTERIM	
INTERIM_DM_EXTENSION14_100DPI	-ADOBE-Interim DM-Medium-I-Normal–20-140-100-100-P-180-DEC-DECMATH_EXTENSION
INTERIM_DM_ITALIC14_100DPI	-ADOBE-Interim DM-Medium-I-Normal–20-140-100-100-P-180-DEC-DECMATH_ITALIC
INTERIM_DM_SYMBOL14_100DPI	-ADOBE-Interim DM-Medium-I-Normal–20-140-100-100-P-180-DEC-DECMATH_SYMBOL
LUBALIN GRAPH	
LUBALINGRAPH_BOOK8_100DPI	-Adobe-ITC Lubalin Graph-Book-R-Normal–11-80-100-100-P-60-ISO8859-1
LUBALINGRAPH_BOOK10_100DPI	-Adobe-ITC Lubalin Graph-Book-R-Normal–14-100-100-100-P-81-ISO8859-1
LUBALINGRAPH_BOOK12_100DPI	-Adobe-ITC Lubalin Graph-Book-R-Normal–17-120-100-100-P-89-ISO8859-1
LUBALINGRAPH_BOOK14_100DPI	-Adobe-ITC Lubalin Graph-Book-R-Normal–19-140-100-100-P-106-ISO8859-1
LUBALINGRAPH_BOOK18_100DPI	-Adobe-ITC Lubalin Graph-Book-R-Normal–24-180-100-100-P-139-ISO8859-1
LUBALINGRAPH_BOOK24_100DPI	-Adobe-ITC Lubalin Graph-Book-R-Normal–33-240-100-100-P-180-ISO8859-1
LUBALINGRAPH_BOOKOBLIQUE8_100DPI	-Adobe-ITC Lubalin Graph-Book-O-Normal–11-80-100-100-P-60-ISO8859-1
LUBALINGRAPH_BOOKOBLIQUE10_100DPI	-Adobe-ITC Lubalin Graph-Book-O-Normal–14-100-100-100-P-82-ISO8859-1
LUBALINGRAPH_BOOKOBLIQUE12_100DPI	-Adobe-ITC Lubalin Graph-Book-O-Normal–19-120-100-100-P-89-ISO8859-1
LUBALINGRAPH_BOOKOBLIQUE14_100DPI	-Adobe-ITC Lubalin Graph-Book-O-Normal–20-140-100-100-P-105-ISO8859-1
LUBALINGRAPH_BOOKOBLIQUE18_100DPI	-Adobe-ITC Lubalin Graph-Book-O-Normal–24-180-100-100-P-140-ISO8859-1
LUBALINGRAPH_BOOKOBLIQUE24_100DPI	-Adobe-ITC Lubalin Graph-Book-O-Normal–33-240-100-100-P-181-ISO8859-1
LUBALINGRAPH_DEMI8_100DPI	-Adobe-ITC Lubalin Graph-Demi-R-Normal–11-80-100-100-P-61-ISO8859-1
LUBALINGRAPH_DEMI10_100DPI	-Adobe-ITC Lubalin Graph-Demi-R-Normal–14-100-100-100-P-85-ISO8859-1
LUBALINGRAPH_DEMI12_100DPI	-Adobe-ITC Lubalin Graph-Demi-R-Normal–17-120-100-100-P-92-ISO8859-1
LUBALINGRAPH_DEMI14_100DPI	-Adobe-ITC Lubalin Graph-Demi-R-Normal–19-140-100-100-P-109-ISO8859-1
LUBALINGRAPH_DEMI18_100DPI	-Adobe-ITC Lubalin Graph-Demi-R-Normal–24-180-100-100-P-144-ISO8859-1
LUBALINGRAPH_DEMI24_100DPI	-Adobe-ITC Lubalin Graph-Demi-R-Normal–33-240-100-100-P-184-ISO8859-1
LUBALINGRAPH_DEMIOBLIQUE8_100DPI	-Adobe-ITC Lubalin Graph-Demi-O-Normal–11-80-100-100-P-62-ISO8859-1
LUBALINGRAPH_DEMIOBLIQUE10_100DPI	-Adobe-ITC Lubalin Graph-Demi-O-Normal–14-100-100-100-P-85-ISO8859-1

(continued on next page)

Table D–2 (Cont.) VMS DECwindows 100 DPI Fonts

File Name	Font Name
LUBALIN GRAPH	
LUBALINGRAPH_DEMIOBLIQUE12_100DPI	-Adobe-ITC Lubalin Graph-Demi-O-Normal–17-120-100-100-P-92-ISO8859-1
LUBALINGRAPH_DEMIOBLIQUE14_100DPI	-Adobe-ITC Lubalin Graph-Demi-O-Normal–19-140-100-100-P-109-ISO8859-1
LUBALINGRAPH_DEMIOBLIQUE18_100DPI	-Adobe-ITC Lubalin Graph-Demi-O-Normal–24-180-100-100-P-144-ISO8859-1
LUBALINGRAPH_DEMIOBLIQUE24_100DPI	-Adobe-ITC Lubalin Graph-Demi-O-Normal–33-240-100-100-P-184-ISO8859-1
MENU	
MENU10_100DPI	-Bigelow & Holmes-Menu-Medium-R-Normal–14-100-100-100-P-77-ISO8859-1
MENU12_100DPI	-Bigelow & Holmes-Menu-Medium-R-Normal–17-120-100-100-P-92-ISO8859-1
NEW CENTURY SCHOOLBOOK	
NEWCENTURYSCHLBK_BOLD8_100DPI	-Adobe-New Century Schoolbook-Bold-R-Normal–11-80-100-100-P-66-ISO8859-1
NEWCENTURYSCHLBK_BOLD10_100DPI	-Adobe-New Century Schoolbook-Bold-R-Normal–14-100-100-100-P-87-ISO8859-1
NEWCENTURYSCHLBK_BOLD12_100DPI	-Adobe-New Century Schoolbook-Bold-R-Normal–17-120-100-100-P-99-ISO8859-1
NEWCENTURYSCHLBK_BOLD14_100DPI	-Adobe-New Century Schoolbook-Bold-R-Normal–20-140-100-100-P-113-ISO8859-1
NEWCENTURYSCHLBK_BOLD18_100DPI	-Adobe-New Century Schoolbook-Bold-R-Normal–25-180-100-100-P-149-ISO8859-1
NEWCENTURYSCHLBK_BOLD24_100DPI	-Adobe-New Century Schoolbook-Bold-R-Normal–34-240-100-100-P-193-ISO8859-1
NEWCENTURYSCHLBK_BOLDITALIC8_100DPI	-Adobe-New Century Schoolbook-Bold-I-Normal–11-80-100-100-P-66-ISO8859-1
NEWCENTURYSCHLBK_BOLDITALIC10_100DPI	-Adobe-New Century Schoolbook-Bold-I-Normal–14-100-100-100-P-88-ISO8859-1
NEWCENTURYSCHLBK_BOLDITALIC12_100DPI	-Adobe-New Century Schoolbook-Bold-I-Normal–17-120-100-100-P-99-ISO8859-1
NEWCENTURYSCHLBK_BOLDITALIC14_100DPI	-Adobe-New Century Schoolbook-Bold-I-Normal–20-140-100-100-P-111-ISO8859-1
NEWCENTURYSCHLBK_BOLDITALIC18_100DPI	-Adobe-New Century Schoolbook-Bold-I-Normal–25-180-100-100-P-148-ISO8859-1
NEWCENTURYSCHLBK_BOLDITALIC24_100DPI	-Adobe-New Century Schoolbook-Bold-I-Normal–34-240-100-100-P-193-ISO8859-1
NEWCENTURYSCHLBK_ITALIC8_100DPI	-Adobe-New Century Schoolbook-Medium-I-Normal–11-80-100-100-P-60-ISO8859-1
NEWCENTURYSCHLBK_ITALIC10_100DPI	-Adobe-New Century Schoolbook-Medium-I-Normal–14-100-100-100-P-81-ISO8859-1
NEWCENTURYSCHLBK_ITALIC12_100DPI	-Adobe-New Century Schoolbook-Medium-I-Normal–17-120-100-100-P-92-ISO8859-1
NEWCENTURYSCHLBK_ITALIC14_100DPI	-Adobe-New Century Schoolbook-Medium-I-Normal–20-140-100-100-P-104-ISO8859-1
NEWCENTURYSCHLBK_ITALIC18_100DPI	-Adobe-New Century Schoolbook-Medium-I-Normal–25-180-100-100-P-136-ISO8859-1
NEWCENTURYSCHLBK_ITALIC24_100DPI	-Adobe-New Century Schoolbook-Medium-I-Normal–34-240-100-100-P-182-ISO8859-1
NEWCENTURYSCHLBK_ROMAN8_100DPI	-Adobe-New Century Schoolbook-Medium-R-Normal–11-80-100-100-P-60-ISO8859-1
NEWCENTURYSCHLBK_ROMAN10_100DPI	-Adobe-New Century Schoolbook-Medium-R-Normal–14-100-100-100-P-82-ISO8859-1
NEWCENTURYSCHLBK_ROMAN12_100DPI	-Adobe-New Century Schoolbook-Medium-R-Normal–17-120-100-100-P-91-ISO8859-1

(continued on next page)

Table D–2 (Cont.) VMS DECwindows 100 DPI Fonts

File Name	Font Name
NEWCENTURYSCHLBK_ROMAN14_100DPI	-Adobe-New Century Schoolbook-Medium-R-Normal–20-140-100-100-P-103-ISO8859-1
NEWCENTURYSCHLBK_ROMAN18_100DPI	-Adobe-New Century Schoolbook-Medium-R-Normal–25-180-100-100-P-136-ISO8859-1
NEWCENTURYSCHLBK_ROMAN24_100DPI	-Adobe-New Century Schoolbook-Medium-R-Normal–34-240-100-100-P-181-ISO8859-1
SOUVENIR	
SOUVENIR_DEMI8_100DPI	-Adobe-ITC Souvenir-Demi-R-Normal–11-80-100-100-P-62-ISO8859-1
SOUVENIR_DEMI10_100DPI	-Adobe-ITC Souvenir-Demi-R-Normal–14-100-100-100-P-90-ISO8859-1
SOUVENIR_DEMI12_100DPI	-Adobe-ITC Souvenir-Demi-R-Normal–17-120-100-100-P-94-ISO8859-1
SOUVENIR_DEMI14_100DPI	-Adobe-ITC Souvenir-Demi-R-Normal–20-140-100-100-P-112-ISO8859-1
SOUVENIR_DEMI18_100DPI	-Adobe-ITC Souvenir-Demi-R-Normal–25-180-100-100-P-149-ISO8859-1
SOUVENIR_DEMI24_100DPI	-Adobe-ITC Souvenir-Demi-R-Normal–34-240-100-100-P-191-ISO8859-1
SOUVENIR_DEMIITALIC8_100DPI	-Adobe-ITC Souvenir-Demi-I-Normal–11-80-100-100-P-67-ISO8859-1
SOUVENIR_DEMIITALIC10_100DPI	-Adobe-ITC Souvenir-Demi-I-Normal–14-100-100-100-P-92-ISO8859-1
SOUVENIR_DEMIITALIC12_100DPI	-Adobe-ITC Souvenir-Demi-I-Normal–17-120-100-100-P-98-ISO8859-1
SOUVENIR_DEMIITALIC14_100DPI	-Adobe-ITC Souvenir-Demi-I-Normal–20-140-100-100-P-115-ISO8859-1
SOUVENIR_DEMIITALIC18_100DPI	-Adobe-ITC Souvenir-Demi-I-Normal–25-180-100-100-P-154-ISO8859-1
SOUVENIR_DEMIITALIC24_100DPI	-Adobe-ITC Souvenir-Demi-I-Normal–34-240-100-100-P-197-ISO8859-1
SOUVENIR_LIGHT8_100DPI	-Adobe-ITC Souvenir-Light-R-Normal–11-80-100-100-P-56-ISO8859-1
SOUVENIR_LIGHT10_100DPI	-Adobe-ITC Souvenir-Light-R-Normal–14-100-100-100-P-79-ISO8859-1
SOUVENIR_LIGHT12_100DPI	-Adobe-ITC Souvenir-Light-R-Normal–17-120-100-100-P-85-ISO8859-1
SOUVENIR_LIGHT14_100DPI	-Adobe-ITC Souvenir-Light-R-Normal–20-140-100-100-P-102-ISO8859-1
SOUVENIR_LIGHT18_100DPI	-Adobe-ITC Souvenir-Light-R-Normal–25-180-100-100-P-135-ISO8859-1
SOUVENIR_LIGHT24_100DPI	-Adobe-ITC Souvenir-Light-R-Normal–34-240-100-100-P-174-ISO8859-1
SOUVENIR_LIGHTITALIC8_100DPI	-Adobe-ITC Souvenir-Light-I-Normal–11-80-100-100-P-59-ISO8859-1
SOUVENIR_LIGHTITALIC10_100DPI	-Adobe-ITC Souvenir-Light-I-Normal–14-100-100-100-P-82-ISO8859-1
SOUVENIR_LIGHTITALIC12_100DPI	-Adobe-ITC Souvenir-Light-I-Normal–17-120-100-100-P-88-ISO8859-1
SOUVENIR_LIGHTITALIC14_100DPI	-Adobe-ITC Souvenir-Light-I-Normal–20-140-100-100-P-104-ISO8859-1
SOUVENIR_LIGHTITALIC18_100DPI	-Adobe-ITC Souvenir-Light-I-Normal–25-180-100-100-P-139-ISO8859-1
SOUVENIR_LIGHTITALIC24_100DPI	-Adobe-ITC Souvenir-Light-I-Normal–34-240-100-100-P-177-ISO8859-1
SYMBOL	
SYMBOL8_100DPI	-Adobe-Symbol-Medium-R-Normal–11-80-100-100-P-61-ADOBE-FONTSPECIFIC
SYMBOL10_100DPI	-Adobe-Symbol-Medium-R-Normal–14-100-100-100-P-85-ADOBE-FONTSPECIFIC
SYMBOL12_100DPI	-Adobe-Symbol-Medium-R-Normal–17-120-100-100-P-95-ADOBE-FONTSPECIFIC

(continued on next page)

Table D–2 (Cont.) VMS DECwindows 100 DPI Fonts

File Name	Font Name
SYMBOL	
SYMBOL14_100DPI	-Adobe-Symbol-Medium-R-Normal–20-140-100-100-P-107-ADOBE-FONTSPECIFIC
SYMBOL18_100DPI	-Adobe-Symbol-Medium-R-Normal–25-180-100-100-P-142-ADOBE-FONTSPECIFIC
SYMBOL24_100DPI	-Adobe-Symbol-Medium-R-Normal–34-240-100-100-P-191-ADOBE-FONTSPECIFIC
TERMINAL	
TERMINAL10_100DPI	-DEC-Terminal-Medium-R-Normal–14-100-100-100-C-8-ISO8859-1
TERMINAL14_100DPI	-Bitstream-Terminal-Medium-R-Normal–20-140-100-100-C-11-ISO8859-1
TERMINAL20_100DPI	-DEC-Terminal-Medium-R-Normal–28-200-100-100-C-16-ISO8859-1
TERMINAL28_100DPI	-Bitstream-Terminal-Medium-R-Normal–40-280-100-100-C-22-ISO8859-1
TERMINAL_BOLD10_100DPI	-DEC-Terminal-Bold-R-Normal–14-100-100-100-C-8-ISO8859-1
TERMINAL_BOLD14_100DPI	-Bitstream-Terminal-Bold-R-Normal–20-140-100-100-C-11-ISO8859-1
TERMINAL_BOLD20_100DPI	-DEC-Terminal-Bold-R-Normal–28-200-100-100-C-16-ISO8859-1
TERMINAL_BOLD28_100DPI	-Bitstream-Terminal-Bold-R-Normal–40-280-100-100-C-22-ISO8859-1
TERMINAL_BOLD_DBLWIDE10_100DPI	-DEC-Terminal-Bold-R-Double Wide–14-100-100-100-C-16-ISO8859-1
TERMINAL_BOLD_DBLWIDE14_100DPI	-Bitstream-Terminal-Bold-R-Double Wide–20-140-100-100-C-22-ISO8859-1
TERMINAL_BOLD_DBLWIDE_DECTECH10_100DPI	-DEC-Terminal-Bold-R-Double Wide–14-100-100-100-C-16-DEC-DECtech
TERMINAL_BOLD_DBLWIDE_DECTECH14_100DPI	-Bitstream-Terminal-Bold-R-Double Wide–20-140-100-100-C-22-DEC-DECtech
TERMINAL_BOLD_DECTECH10_100DPI	-DEC-Terminal-Bold-R-Normal–14-100-100-100-C-8-DEC-DECtech
TERMINAL_BOLD_DECTECH14_100DPI	-Bitstream-Terminal-Bold-R-Normal–20-140-100-100-C-11-DEC-DECtech
TERMINAL_BOLD_DECTECH20_100DPI	-DEC-Terminal-Bold-R-Normal–28-200-100-100-C-16-DEC-DECtech
TERMINAL_BOLD_DECTECH28_100DPI	-Bitstream-Terminal-Bold-R-Normal–40-280-100-100-C-22-DEC-DECtech
TERMINAL_BOLD_NARROW10_100DPI	-DEC-Terminal-Bold-R-Narrow–14-100-100-100-C-6-ISO8859-1
TERMINAL_BOLD_NARROW14_100DPI	-Bitstream-Terminal-Bold-R-Narrow–20-140-100-100-C-7-ISO8859-1
TERMINAL_BOLD_NARROW20_100DPI	-DEC-Terminal-Bold-R-Narrow–28-200-100-100-C-12-ISO8859-1
TERMINAL_BOLD_NARROW28_100DPI	-Bitstream-Terminal-Bold-R-Narrow–40-280-100-100-C-14-ISO8859-1
TERMINAL_BOLD_NARROW_DECTECH10_100DPI	-DEC-Terminal-Bold-R-Narrow–14-100-100-100-C-6-DEC-DECtech
TERMINAL_BOLD_NARROW_DECTECH14_100DPI	-Bitstream-Terminal-Bold-R-Narrow–20-140-100-100-C-7-DEC-DECtech
TERMINAL_BOLD_NARROW_DECTECH20_100DPI	-DEC-Terminal-Bold-R-Narrow–28-200-100-100-C-12-DEC-DECtech
TERMINAL_BOLD_NARROW_DECTECH28_100DPI	-Bitstream-Terminal-Bold-R-Narrow–40-280-100-100-C-14-DEC-DECtech
TERMINAL_BOLD_WIDE10_100DPI	-DEC-Terminal-Bold-R-Wide–14-100-100-100-C-12-ISO8859-1
TERMINAL_BOLD_WIDE14_100DPI	-Bitstream-Terminal-Bold-R-Narrow–20-140-100-100-C-14-ISO8859-1
TERMINAL_BOLD_WIDE_DECTECH10_100DPI	-DEC-Terminal-Bold-R-Wide–14-100-100-100-C-12-DEC-DECtech
TERMINAL_BOLD_WIDE_DECTECH14_100DPI	-Bitstream-Terminal-Bold-R-Narrow–20-140-100-100-C-14-DEC-DECtech
TERMINAL_DBLWIDE10_100DPI	-DEC-Terminal-Medium-R-Double Wide–14-100-100-100-C-16-ISO8859-1
TERMINAL_DBLWIDE14_100DPI	-Bitstream-Terminal-Medium-R-Double Wide–20-140-100-100-C-22-ISO8859-1

(continued on next page)

Table D–2 (Cont.) VMS DECwindows 100 DPI Fonts

File Name	Font Name
TERMINAL	
TERMINAL_DBLWIDE_DECTECH10_100DPI	-DEC-Terminal-Medium-R-Double Wide–14-100-100-100-C-16-DEC-DECtech
TERMINAL_DBLWIDE_DECTECH14_100DPI	-Bitstream-Terminal-Medium-R-Double Wide–20-140-100-100-C-22-DEC-DECtech
TERMINAL_DECTECH10_100DPI	-DEC-Terminal-Medium-R-Normal–14-100-100-100-C-8-DEC-DECtech
TERMINAL_DECTECH14_100DPI	-Bitstream-Terminal-Medium-R-Normal–20-140-100-100-C-11-DEC-DECtech
TERMINAL_DECTECH20_100DPI	-DEC-Terminal-Medium-R-Normal–28-200-100-100-C-16-DEC-DECtech
TERMINAL_DECTECH28_100DPI	-Bitstream-Terminal-Medium-R-Normal–40-280-100-100-C-22-DEC-DECtech
TERMINAL_NARROW10_100DPI	-DEC-Terminal-Medium-R-Narrow–14-100-100-100-C-6-ISO8859-1
TERMINAL_NARROW14_100DPI	-Bitstream-Terminal-Medium-R-Narrow–20-140-100-100-C-7-ISO8859-1
TERMINAL_NARROW20_100DPI	-DEC-Terminal-Medium-R-Narrow–28-200-100-100-C-12-ISO8859-1
TERMINAL_NARROW28_100DPI	-Bitstream-Terminal-Medium-R-Narrow–40-280-100-100-C-14-ISO8859-1
TERMINAL_NARROW_DECTECH10_100DPI	-DEC-Terminal-Medium-R-Narrow–14-100-100-100-C-6-DEC-DECtech
TERMINAL_NARROW_DECTECH14_100DPI	-Bitstream-Terminal-Medium-R-Narrow–20-140-100-100-C-7-DEC-DECtech
TERMINAL_NARROW_DECTECH20_100DPI	-DEC-Terminal-Medium-R-Narrow–28-200-100-100-C-12-DEC-DECtech
TERMINAL_NARROW_DECTECH28_100DPI	-Bitstream-Terminal-Medium-R-Narrow–40-280-100-100-C-14-DEC-DECtech
TERMINAL_WIDE10_100DPI	-DEC-Terminal-Medium-R-Wide–14-100-100-100-C-12-ISO8859-1
TERMINAL_WIDE14_100DPI	-Bitstream-Terminal-Medium-R-Wide–20-140-100-100-C-14-ISO8859-1
TERMINAL_WIDE_DECTECH10_100DPI	-DEC-Terminal-Medium-R-Wide–14-100-100-100-C-12-DEC-DECtech
TERMINAL_WIDE_DECTECH14_100DPI	-Bitstream-Terminal-Medium-R-Wide–20-140-100-100-C-14-DEC-DECtech
TIMES	
TIMES_BOLD8_100DPI	-Adobe-Times-Bold-R-Normal–11-80-100-100-P-57-ISO8859-1
TIMES_BOLD10_100DPI	-Adobe-Times-Bold-R-Normal–14-100-100-100-P-76-ISO8859-1
TIMES_BOLD12_100DPI	-Adobe-Times-Bold-R-Normal–17-120-100-100-P-88-ISO8859-1
TIMES_BOLD14_100DPI	-Adobe-Times-Bold-R-Normal–20-140-100-100-P-100-ISO8859-1
TIMES_BOLD18_100DPI	-Adobe-Times-Bold-R-Normal–25-180-100-100-P-132-ISO8859-1
TIMES_BOLD24_100DPI	-Adobe-Times-Bold-R-Normal–34-240-100-100-P-177-ISO8859-1
TIMES_BOLDITALIC8_100DPI	-Adobe-Times-Bold-I-Normal–11-80-100-100-P-57-ISO8859-1
TIMES_BOLDITALIC10_100DPI	-Adobe-Times-Bold-I-Normal–14-100-100-100-P-77-ISO8859-1
TIMES_BOLDITALIC12_100DPI	-Adobe-Times-Bold-I-Normal–17-120-100-100-P-86-ISO8859-1
TIMES_BOLDITALIC14_100DPI	-Adobe-Times-Bold-I-Normal–20-140-100-100-P-98-ISO8859-1
TIMES_BOLDITALIC18_100DPI	-Adobe-Times-Bold-I-Normal–25-180-100-100-P-128-ISO8859-1
TIMES_BOLDITALIC24_100DPI	-Adobe-Times-Bold-I-Normal–34-240-100-100-P-170-ISO8859-1
TIMES_ITALIC8_100DPI	-Adobe-Times-Medium-I-Normal–11-80-100-100-P-52-ISO8859-1
TIMES_ITALIC10_100DPI	-Adobe-Times-Medium-I-Normal–14-100-100-100-P-73-ISO8859-1
TIMES_ITALIC12_100DPI	-Adobe-Times-Medium-I-Normal–17-120-100-100-P-84-ISO8859-1

(continued on next page)

Table D–2 (Cont.) VMS DECwindows 100 DPI Fonts

File Name	Font Name
TIMES	
TIMES_ITALIC14_100DPI	-Adobe-Times-Medium-I-Normal–20-140-100-100-P-94-ISO8859-1
TIMES_ITALIC18_100DPI	-Adobe-Times-Medium-I-Normal–25-180-100-100-P-125-ISO8859-1
TIMES_ITALIC24_100DPI	-Adobe-Times-Medium-I-Normal–34-240-100-100-P-168-ISO8859-1
TIMES_ROMAN8_100DPI	-Adobe-Times-Medium-R-Normal–11-80-100-100-P-54-ISO8859-1
TIMES_ROMAN10_100DPI	-Adobe-Times-Medium-R-Normal–14-100-100-100-P-74-ISO8859-1
TIMES_ROMAN12_100DPI	-Adobe-Times-Medium-R-Normal–17-120-100-100-P-84-ISO8859-1
TIMES_ROMAN14_100DPI	-Adobe-Times-Medium-R-Normal–20-140-100-100-P-96-ISO8859-1
TIMES_ROMAN18_100DPI	-Adobe-Times-Medium-R-Normal–25-180-100-100-P-125-ISO8859-1
TIMES_ROMAN24_100DPI	-Adobe-Times-Medium-R-Normal–34-240-100-100-P-170-ISO8859-1

Index

How to Order Additional Documentation

Technical Support

If you need help deciding which documentation best meets your needs, call 800-343-4040 before placing your electronic, telephone, or direct mail order.

Electronic Orders

To place an order at the Electronic Store, dial 800-DEC-DEMO (800-332-3366) using a 1200- or 2400-baud modem. If you need assistance using the Electronic Store, call 800-DIGITAL (800-344-4825).

Telephone and Direct Mail Orders

Your Location	Call	Contact
Continental USA, Alaska, or Hawaii	800-DIGITAL	Digital Equipment Corporation P.O. Box CS2008 Nashua, New Hampshire 03061
Puerto Rico	809-754-7575	Local DIGITAL subsidiary
Canada	800-267-6215	Digital Equipment of Canada Attn: DECdirect Operations KAO2/2 P.O. Box 13000 100 Herzberg Road Kanata, Ontario, Canada K2K 2A6
International	————	Local DIGITAL subsidiary or approved distributor
Internal[1]	————	SDC Order Processing - WMO/E15 *or* Software Distribution Center Digital Equipment Corporation Westminster, Massachusetts 01473

[1]For internal orders, you must submit an Internal Software Order Form (EN-01740-07).

Reader's Comments

VMS DECwindows
Guide to Xlib Programming:
MIT C Binding
AA–MG24A–TE

Please use this postage-paid form to comment on this manual. If you require a written reply to a software problem and are eligible to receive one under Software Performance Report (SPR) service, submit your comments on an SPR form.

Thank you for your assistance.

I rate this manual's:	Excellent	Good	Fair	Poor
Accuracy (software works as manual says)	☐	☐	☐	☐
Completeness (enough information)	☐	☐	☐	☐
Clarity (easy to understand)	☐	☐	☐	☐
Organization (structure of subject matter)	☐	☐	☐	☐
Figures (useful)	☐	☐	☐	☐
Examples (useful)	☐	☐	☐	☐
Index (ability to find topic)	☐	☐	☐	☐
Page layout (easy to find information)	☐	☐	☐	☐

I would like to see more/less ______________________________

What I like best about this manual is ______________________________

What I like least about this manual is ______________________________

I found the following errors in this manual:

Page Description

Additional comments or suggestions to improve this manual:

I am using **Version** ______ of the software this manual describes.

Name/Title ______________________ Dept. ____________
Company ______________________ Date ____________
Mailing Address ______________________________
______________________ Phone ____________

-— Do Not Tear - Fold Here and Tape

digital™

No Postage
Necessary
if Mailed
in the
United States

BUSINESS REPLY MAIL
FIRST CLASS PERMIT NO. 33 MAYNARD MASS.

POSTAGE WILL BE PAID BY ADDRESSEE

DIGITAL EQUIPMENT CORPORATION
Corporate User Publications—Spit Brook
ZK01–3/J35 110 SPIT BROOK ROAD
NASHUA, NH 03062-9987

-— Do Not Tear - Fold Here

Cut Along Dotted Line

Reader's Comments

VMS DECwindows
Guide to Xlib Programming:
MIT C Binding
AA–MG24A–TE

Please use this postage-paid form to comment on this manual. If you require a written reply to a software problem and are eligible to receive one under Software Performance Report (SPR) service, submit your comments on an SPR form.

Thank you for your assistance.

I rate this manual's:	Excellent	Good	Fair	Poor
Accuracy (software works as manual says)	☐	☐	☐	☐
Completeness (enough information)	☐	☐	☐	☐
Clarity (easy to understand)	☐	☐	☐	☐
Organization (structure of subject matter)	☐	☐	☐	☐
Figures (useful)	☐	☐	☐	☐
Examples (useful)	☐	☐	☐	☐
Index (ability to find topic)	☐	☐	☐	☐
Page layout (easy to find information)	☐	☐	☐	☐

I would like to see more/less ______________________________

What I like best about this manual is ______________________________

What I like least about this manual is ______________________________

I found the following errors in this manual:

Page Description

Additional comments or suggestions to improve this manual:

I am using **Version** ______ of the software this manual describes.

Name/Title ______________________________ Dept. ____________

Company ______________________________ Date ____________

Mailing Address ______________________________

______________________________ Phone ____________

— Do Not Tear - Fold Here and Tape

digital™

No Postage
Necessary
if Mailed
in the
United States

BUSINESS REPLY MAIL
FIRST CLASS PERMIT NO. 33 MAYNARD MASS.

POSTAGE WILL BE PAID BY ADDRESSEE

DIGITAL EQUIPMENT CORPORATION
Corporate User Publications—Spit Brook
ZK01-3/J35 110 SPIT BROOK ROAD
NASHUA, NH 03062-9987

— Do Not Tear - Fold Here

Cut Along Dotted Line