AIX Operating System
IBM AIX X-Windows
Programmer's Reference

Document Number SC23-2118-02

Copyright IBM Corp. 1985, 1991

AIX Operating System
IBM AIX X-Windows

Programmer's Reference

Document Number SC23-2118-02

Copyright IBM Corp. 1985, 1991

Edition Notice

Third Edition (March 1991)

This edition applies to Version 2.1 of IBM AIX/RT X-Windows and Versions 1.1 and 1.2 of IBM AIX PS/2 X-Windows. Make sure you are using the correct edition for the level of the product.

Order publications through your IBM representative or the IBM branch office serving your locality. Publications are not stocked at the address given below.

A form for reader's comments appears at the back of this publication. If the form has been removed, address your comments to:

IBM Corporation, Department 52QA MS 911
Neighborhood Road
Kingston, NY 12401
U.S.A.

When you send information to IBM, you grant IBM a nonexclusive right to use or distribute the information in any way it believes appropriate without incurring any obligation to you.

¦ Adobe Systems, Inc. 1984, 1987
¦ Digital Equipment Corporation 1985, 1988
¦ Massachusetts Institute of Technology 1985, 1988

¦ Copyright International Business Machines Corporation 1985, 1991. All rights reserved.
Note to U.S. Government Users -- Documentation related to restricted rights -- Use, duplication or disclosure is subject to restrictions set forth in GSA ADP Schedule Contract with IBM Corp.

Notices

References in this publication to IBM products, programs, or services do not imply that IBM intends to make these available in all countries in which IBM operates. Any reference to an IBM product, program, or service is not intended to state or imply that only IBM's product, program, or service may be used. Any functionally equivalent product, program, or service that does not infringe any of IBM's intellectual property rights or other legally protectible rights may be used instead of the IBM product, program, or service. Evaluation and verification of operation in conjunction with other products, programs, or services, except those expressly designated by IBM, are the user's responsibility.

IBM may have patents or pending patent applications covering subject matter in this document. The furnishing of this document does not give you any license to these patents. You can send license inquiries, in writing, to the IBM Director of Commercial Relations, IBM Corporation, Purchase, NY 10577.

Subtopics

Trademarks and Acknowledgments

¦ Copyright IBM Corp. 1985, 1991

Trademarks and Acknowledgments

The following trademarks apply to this book:

AIX is a registered trademark of International Business Machine Corporation.

DEC is a trademark of Digital Equipment Corporation

IBM is a registered trademark of International Business Machine Corporation.

INed is a registered trademark of INTERACTIVE Systems Corporation

PS/2 is a registered trademark of International Business Machine Corporation.

RT is a registered trademark of International Business Machine Corporation.

S/370 is a trademark of International Business Machines Corporation

UNIX is a registered trademark of UNIX System Laboratories, Inc. i the USA and other countries.

VT100 and VT102 are trademarks of Digital Equipment Corporation

X-Windows is a trademark of Massachusetts Institute of Technology

Portions of the code and documentation described in this book wer developed at the Electrical Engineering and Computer Sciences Department at the Berkeley Campus of the University of California under the auspices of the Regents of the University of California.

Portions of the code and documentation described in this book wer derived from code and documentation developed by Adobe Systems, Inc.

Portions of the code and documentation described in this book wer derived from code and documentation developed by Massachusetts Institute of Technology, Cambridge, Massachusetts, and Digital Equipment Corporation, Maynard, Massachusetts.

¦ Copyright IBM Corp. 1985, 1991

About This Book

This book describes the IBM AIX X-Windows licensed program and provides detailed reference information on X-Windows library functions, FORTRAN subroutines, protocols, and extensions.

Subtopics

Who Should Read This Book
How to Use This Book
Related Publications

¦ Copyright IBM Corp. 1985, 1991

Who Should Read This Book

This book is intended for anyone using X-Windows macros and functions. It assumes that you have a basic understanding of a graphics window system and the C or FORTRAN programming language.

Before you use X-Windows, you must have the latest level of the AIX Operating System installed. You must also have the level of the IBM AIX X-Windows licensed program appropriate for the type of hardware you are using.

¦ Copyright IBM Corp. 1985, 1991

How to Use This Book

First read Chapter 1, "Using X-Windows," which describes the X-Windows licensed program and how it works.

Then read the remaining chapters in any order your want.

For information on terms unique to X-Windows, see the glossary at the back of this book.

Subtopics
Highlighting
Naming and Syntax Conventions
For Japanese Locale Users

¦ Copyright IBM Corp. 1985, 1991

Highlighting

New Terms	Each time a new term is introduced, its first occurrence is printed in this type style (for example, "the AIX Operating System ***file system***"). These words also are defined in the glossary.
System parts	The names for keys, commands, file names, procedure names, and structure names are printed in this type style (for example, the **xinit** and **xterm** commands).
Examples and information you type	Names of files that have been created for examples in this book are printed in **monospace black**, for example, **AIX Shell**. Text that you type or that is displayed on your screen is printed also in **monospace black**, for example, Select **Move** from the window manager menu.
Variable information	Arguments or user-supplied variables are printed in this type style.

¦ Copyright IBM Corp. 1985, 1991

Naming and Syntax Conventions

Throughout this book, a number of naming conventions and syntax conventions have been followed. These conventions are intended to make the syntax of the functions more predictable.

The major naming conventions are:

Mixed case is used for external symbols to differentiate betwee X-Windows symbols and user symbols. User macros are in all uppercase and variables are in all lowercase.

Xlib functions begin with a capital X.

Procedure names and symbols begin with a capital letter

User-visible data structures begin with a capital X. Anything that user might dereference begins with a capital X.

Macros and other symbols do not begin with a capital X. T distinguish them from all user symbols, each word in the macro is capitalized.

All elements of or variables in a data structure are in lowercase Compound words, where needed, are constructed with underscores (_).

The *display* argument is always first in the argument list.

All resource objects occur at the beginning of the argument list immediately after the *display* variable.

When a graphics context (GC) is present with another type of resourc (most commonly, a drawable), GC occurs in the argument list after the other resource. Drawables outrank all other resources.

Source arguments always precede the destination arguments in the argument list.

The *x* argument always precedes the *y* argument in the argument list.

The *width* argument always precedes the *height* argument in the argument list.

When the *x*, *y*, *width* and *height* arguments are used together, the *x* and *y* arguments always precede the *width* and *height* arguments.

When an *array* occurs in an argument list accompanied with a *count* (number of elements in the *array*), the *array* always precedes the *count* in the argument list.

When a *mask* is accompanied with a structure, it always precedes the pointer to the structure in the argument list.

Arguments are explained after the variable list is defined. The explanations for arguments that the user passes begin with the word *specifies*. The explanations for arguments that are returned to the user begin with the word *returns*. A general description of the procedure, if required, follows the arguments.

¦ Copyright IBM Corp. 1985, 1991

PREFACE.2.3 For Japanese Locale Users

The X-Windows software is the prime enabling software for multibyte character set (MBCS). Like all standard programs in the AIX system, all commands, subcommands and option flags must be input in ASCII. The user must have the keyboard in the ASCII entry mode for any instructions being made to the X-Windows software. Once X-Windows is operating, the user can switch the keyboard to any of the Japanese text entry modes. For specific information regarding MBCS, see *AIX Guide to Multibyte Character Set (MBCS) Support* and "Programming for an MBCS Environment" in *AIX Operating System Programming Tool and Interfaces*.

¦ Copyright IBM Corp. 1985, 1991

Related Publications

For additional information, you may want to refer to the following publications:

AIX C Language Reference, SC23-2058, describes the C programming language and contains reference information for writing programs in C language that run on the AIX Operating System.

AIX C Language User's Guide, SC23-2057, describes how to develop, link, and execute C language programs. This book also describes the operating dependencies of C language and shows how to use C language-related software utilities and other program development tools.

AIX Commands Reference, SC23-2292 (Vol. 1) and SC23-2184 (Vol. 2), lists and describes the AIX/370 and AIX PS/2 Operating System commands.

AIX Guide to Multibyte Character Set (MBCS) Support, GC23-2333, explains the basic concepts of AIX multibyte character set support and refers to other AIX books that contain more detailed information.

AIX Messages Reference, SC23-2294, lists messages displayed by the AIX Operating System and explains how to respond to them.

AIX Programming Tools and Interfaces, SC23-2304, describes the programming environment of the AIX Operating System and includes information about operating system tools that are used to develop, compile, and debug programs.

AIX TCP/IP User's Guide, SC23-2309, describes the features of TCP/IP and shows how to install and customize the program. It includes reference information on TCP/IP commands that are used to transfer files, manage the network, and log into remote systems.

AIX Technical Reference, SC23-2300 (Vol. 1) and SC23-2301 (Vol. 2), describes the system calls and subroutines a programmer uses to write application programs. This book also provides information about the AIX Operating System file system, special files, miscellaneous files, and the writing of device drivers.

AIX VS FORTRAN Reference, SC23-2050, describes the FORTRAN programming language as implemented on AIX RT, AIX PS/2, and AIX/370. This book describes all of the standard features of VS FORTRAN as well as the enhanced functions and capabilities incorporated into IBM AIX VS FORTRAN.

AIX VS FORTRAN User's Guide, SC23-2049, shows how to develop and execute FORTRAN programs on AIX RT, AIX PS/2, and AIX/370. This book also explains how to compile and execute programs that contain sections of code written in the VS Pascal and C programming languages.

AIX X-Windows User's Guide, SC23-2017, describes the X-Windows licensed program and shows how to start, run, install, and customize this program.

AIX PS/2 Keyboard Description and Character Reference, SC23-2037, describes the characters and keyboards supported by the AIX PS/2 Operating System. This book also provides information on keyboard

¦ Copyright IBM Corp. 1985, 1991

position codes, keyboard states, control code points, code-sequence processing, and non-spacing character sequences.

Installing and Customizing the AIX PS/2 Operating System, SC23-2290, provides step-by-step instructions for installing the AIX PS/2 Operating System and related programs. This book also shows how to customize the operating system to suit the user's specific needs and work environment.

Managing the AIX Operating System, SC23-2293, describes such system-management tasks as adding and deleting user IDs, creating and mounting file systems, backing up the system, repairing file system damage, and setting up an electronic mail system and other networking facilities.

Using the AIX Operating System, SC23-2291, shows the beginning user how to use AIX Operating System commands to do such basic tasks as log in and out of the system, display and print files, and set and change passwords. It includes information for intermediate to advanced users about how to use communication and networking facilities and write shell procedures.

¦ Copyright IBM Corp. 1985, 1991

Table of Contents

¦ Copyright IBM Corp. 1985, 1991

¦ Copyright IBM Corp. 1985, 1991

¦ Copyright IBM Corp. 1985, 1991

¦ Copyright IBM Corp. 1985, 1991

¦ Copyright IBM Corp. 1985, 1991

¦ Copyright IBM Corp. 1985, 1991

¦ Copyright IBM Corp. 1985, 1991

¦ Copyright IBM Corp. 1985, 1991

¦ Copyright IBM Corp. 1985, 1991

¦ Copyright IBM Corp. 1985, 1991

¦ Copyright IBM Corp. 1985, 1991

¦ Copyright IBM Corp. 1985, 1991

¦ Copyright IBM Corp. 1985, 1991

¦ Copyright IBM Corp. 1985, 1991

¦ Copyright IBM Corp. 1985, 1991

¦ Copyright IBM Corp. 1985, 1991

¦ Copyright IBM Corp. 1985, 1991

¦ Copyright IBM Corp. 1985, 1991

¦ Copyright IBM Corp. 1985, 1991

¦ Copyright IBM Corp. 1985, 1991

¦ Copyright IBM Corp. 1985, 1991

¦ Copyright IBM Corp. 1985, 1991

¦ Copyright IBM Corp. 1985, 1991

¦ Copyright IBM Corp. 1985, 1991

¦ Copyright IBM Corp. 1985, 1991

¦ Copyright IBM Corp. 1985, 1991

¦ Copyright IBM Corp. 1985, 1991

1.0 Chapter 1. Using X-Windows

Subtopics

¦ Copyright IBM Corp. 1985, 1991

1.1 CONTENTS

¦ Copyright IBM Corp. 1985, 1991

1.2 About This Chapter

This chapter is organized by X-Windows programming tasks. This chapter describes how X-Windows works. **Xlib** functions related to a given programming task are listed after the description of the tasks.

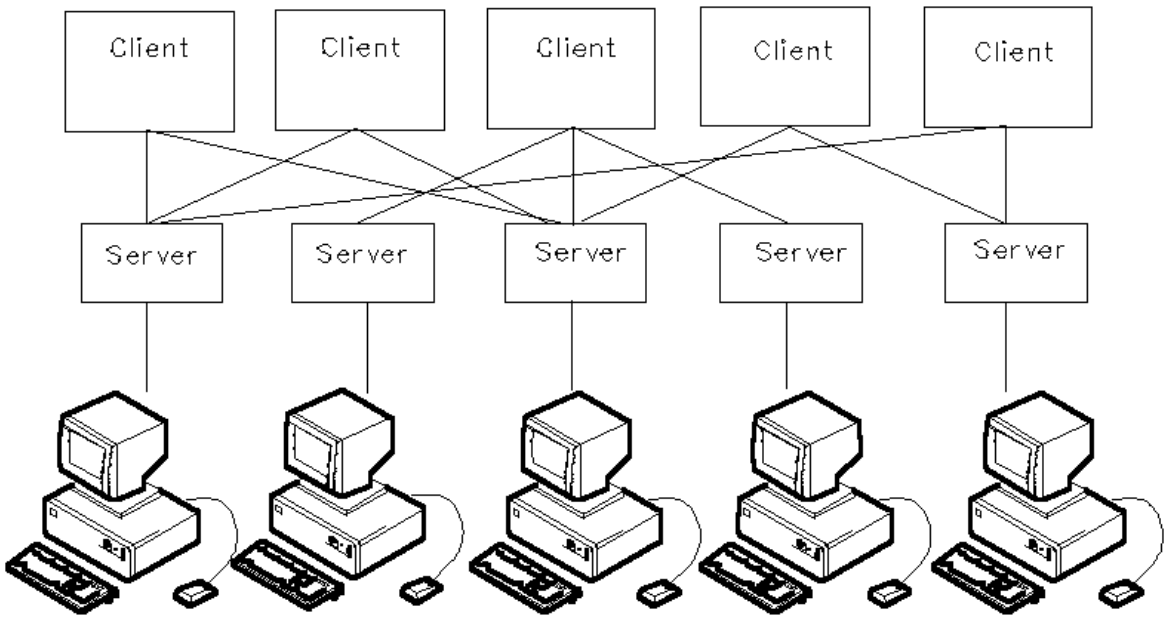
Detailed information about **Xlib** functions can be found in Chapter 2, "X-Windows Xlib Functions." Detailed information about **FXlib** functions can be found in Chapter 3, "FXlib Functions." (The FORTRAN bindings are supported on the RT only.)

¦ Copyright IBM Corp. 1985, 1991

1.3 Overview

X-Windows is a network-transparent windowing system that runs under the AIX Operating System. X-Windows runs on systems with bitmapped display terminals. The X Server distributes user input to and accepts output requests from various client programs located either on the same system or elsewhere in a network. **Xlib** is a **C** language subroutine library that client programs use to interface with the windowing system.

The application program you create is the ***client*** part of a client-server relationship; you write the program and the X Server provides it independence from the hardware. There is one X Server for each virtual terminal that runs X-Windows. In this publication, the term ***display*** refers to a logical virtual terminal with its associated keyboard, locator, and server unless it is explicitly stated otherwise. The following diagram shows the client-X Server-display relationship:



Each client can interact with many X Servers, and each X Server can interact with many clients.

The programming interface is based on a network protocol that allows your program (client) to efficiently interact with displays connected to other processors in a network.

Subtopics
1.3.1 Defining Windows
1.3.2 Compiling X Programs

¦ Copyright IBM Corp. 1985, 1991

1.3.1 Defining Windows

X-Windows supports one screen containing overlapping windows or subwindows. A screen is a physical monitor, which can be color or black and white, and hardware.

The windows in an X Server are arranged in a strict hierarchy. At the top of the hierarchy is the root window, which covers the display screen. The root window is partially or completely covered by child windows. All windows, except the root window, have parent windows. There is usually at least one window per application program. Child windows can, in turn, have their own child window. In this way, an application program can create a tree of arbitrary depth on the screen.

A child window may be larger than its parent. Part or all of the child window may extend beyond the boundaries of the parent. However, all output to a window is clipped by the boundaries of its parent window. If several children of a window have overlapping locations, one of the children is considered to be on top of or raised over the others, obscuring them. Output to areas covered by other windows is suppressed by the window system. If a window is obscured by a second window, the window obscures only those ancestors of the second window, which are also ancestors of the first window.

A window has a border of zero or more pixels in ***width***, which can be any pattern or solid color. A window usually, but not always, has a background pattern that is repainted by the window system when the window is uncovered. Each window has its own coordinate system. Child windows obscure the parent window unless the child windows have no background. Graphic operations in the parent window are usually clipped by the child windows.

Input from X-Windows takes the form of events. Events may be side effects of a command (for example, restacking windows generates exposure events) or completely asynchronous (for example, the keyboard). A client program asks to be informed of events. X-Windows never sends events that a program did not ask for.

X-Windows does not take responsibility for the contents of windows. When part or all of a window is hidden and then brought back onto the screen, its contents may be lost, and the client program is notified by an exposure event that part or all of the window needs to be repainted. Programs must be prepared to regenerate the contents of windows on demand.

X-Windows also provides off-screen storage of graphics objects, called ***pixmaps***. Single plane (depth 1) pixmaps are sometimes referred to as ***bitmaps***. Bitmaps can be used in most graphics functions interchangeably with windows and are used in various graphics operations to define patterns, also called ***tiles***. Windows and pixmaps together are referred to as ***drawables***.

Most of the functions in **Xlib** only add requests to an output buffer. These requests execute asynchronously later on the X Server, the display server. Functions that return values of information stored in the server do not return; these functions ***block*** until an explicit reply is received or an error occurs. If a nonblocking call results in an error, the error is generally not reported by a call to an optional error handler until some later blocking call is made.

If X-Windows does not want a request to execute asynchronously, a client

¦ Copyright IBM Corp. 1985, 1991

can follow the request with a call to **XSync**, which blocks until all previously buffered asynchronous events have been sent and acted upon. The output buffer is always flushed by a call to any function that returns a value or waits for input (for example, **XPending**, **XNextEvent**, **XWindowEvent**, **XFlush**, or **XSync**).

Many **Xlib** functions return an integer resource ID that allows you to refer to objects stored on the X Server. These objects can be of type **Window**, **Font**, **Pixmap**, **Bitmap**, **Cursor**, and **GContext**, as defined in the file **<X11/X.h>**. (1) These resources are created by user requests and are destroyed or freed by user requests or by close functions. Most of these resources can be shared by applications. Windows are manipulated explicitly by window manager programs. Fonts and cursors are typically shared automatically since the X Server treats fonts specially, loading and unloading font storage as needed.

Some functions return *status* which is an integer error indication. If the function fails, *status* will be 0. If the function returns a status of 0, the function did not update the return parameters. Because C language does not provide multiple return values, many functions must return their results by writing into client-passed storage. Any pointer that is used to return a value is designated by the *_return* suffix as part of its name. All other pointers passed to these functions are used for reading only. By default, errors are handled either by a standard library function or by a library function that you provide. Functions that return pointers to strings return NULL pointers if the string does not exist.

Input events, for example, key pressed events or mouse moved events, arrive asynchronously from the server and are queued until they are requested by a call to **XNextEvent** or **XWindowEvent**. In addition, some of the library functions, such as **XResizeWindow** and **XRaiseWindow**, generate exposure events or requests to repaint sections of a window that do not have valid contents. These events also arrive asynchronously, but the client may wish to wait for them explicitly by calling **XSync** after calling a function that may generate exposure events.

(1) The <> has the meaning defined by the **#include** statement of the C compiler and is a file relative to a well known directory. On the AIX Operating System this is **/usr/include**.

¦ Copyright IBM Corp. 1985, 1991

1.3.2 Compiling X Programs

The following compiler command can be used to build your program:

cc {*compiler options***} -o***sample sample***.c -lX11 {-lsock -lbsd}**

In this example, ***sample*.c** is the name of your C language source program, *sample* is the name of your executable C program and **lX11** is the X-Windows subroutine library (**/usr/lib/libX11.a**). Some releases of AIX require **libsock.a** and **libbsd.a**.

The compiler definitions for structures, parameters, error codes, and data types are located in the **/usr/include/X11/Xlib.h** and **/usr/include/X11/X.h** files. You must include this file in any program that uses X-Windows subroutines. To do so, insert the following statement early in your program:

```
  #include <X11/Xlib.h>   /* also includes X11/X.h  */
```

¦ Copyright IBM Corp. 1985, 1991

1.4 Using Display Functions

Before a program can use a display, the program must establish a connection to the X Server driving the display. The ***X Server*** is a display server.

Subtopics
1.4.1 Opening a Display
1.4.2 Closing the Display

¦ Copyright IBM Corp. 1985, 1991

1.4.1 Opening a Display

To open a connection to the X Server controlling a specified display, use **XOpenDisplay**. This function returns a display structure that serves as the connection to the X Server. This display structure contains information about the X Server and connects the specified hardware display to the server through **TCP** or **UNIX** domain sockets. If the *hostname* is a host system name and a colon (:) separates the host name and display number, **XOpenDisplay** connects the host and the display using **TCP** sockets. If the *hostname* does not exist or is **UNIX** and a colon (:) separates it from the display number, **XOpenDisplay** connects the host and the display using **UNIX** domain sockets.

A single server can support any or all of these transport mechanisms simultaneously.

The display name or **DISPLAY** environment variable is a string that has the format:

```
hostname:number.screen
```

hostname Specifies the name of the host system where the display is physically attached. The *hostname* should be followed by a colon (:).

number Specifies the *number* of the display server on that host machine. The display number can be followed by a period (.).

screen Specifies the number of the *screen* on that host server. Multiple screens can be connected to or controlled by a single X Server. (2) The screen sets an internal variable that can be accessed with the **DefaultScreen** macro or the **XDefaultScreen** function.

For example, the following would specify screen 0 display 2 on the system named **Dave**:

```
Dave:2.0
```

Applications should not directly modify any part of the **Display** and **Screen** data structures. These structures should be considered read-only by the user, but they can be changed by some functions or display macros.

(2) IBM AIX X-Windows supports only one screen.

Subtopics
1.4.1.1 Related Functions

¦ Copyright IBM Corp. 1985, 1991

1.4.1.1 Related Functions

Display Macro
XOpenDispla
XNoO
XFre

¦ Copyright IBM Corp. 1985, 1991

1.4.2 Closing the Display

When the X Server connection to a client is closed, either by an explicit call to **XCloseDisplay** or by a process that exits, the X Server performs the following operations automatically:

Disowns all selections owned by the client. See **XSetSelectionOwner** in Chapter 2, "X-Windows Xlib Functions."

Performs an **XUngrabPointer** and **XUngrabKeyboard** if the client application has actively grabbed the pointer or the keyboard. See **XUngrabPointer** and **XUngrabKeyboard** in Chapter 2, "X-Windows Xlib Functions."

Performs an **XUngrabServer** if the client has grabbed the server. See **XUngrabServer** in Chapter 2, "X-Windows Xlib Functions."

Releases all passive grabs made by the client application

Marks all resources (including colormap entries) allocated by th client application as permanent or temporary, depending on whether the *close_mode* argument is **RetainPermanent** or **RetainTemporary**. However, this does not prevent other client applications from explicitly destroying the resources. See **XSetCloseDownMode** in Chapter 2, "X-Windows Xlib Functions."

When the X Server connection to a client is closed in the **DestroyAll** mode, the X Server destroys all of the resources of the client application as follows:

Examines each window in the client save-set to determine if th save-set window is an inferior or subwindow of a window created by the client. (The save-set is a list of other client windows that are referred to as ***save-set*** windows.) If so, the X Server reparents the save-set window to the closest ancestor so that the save-set window is not an inferior of a window created by the client.

Performs a **MapWindow** request on the save-set window if the save-set window is unmapped. The X Server performs a **MapWindow** request even if the save-set window is not an inferior of a window created by the client.

Examines each window in the client save-set and destroys all window created by the client.

Performs the appropriate free request on each non-window resourc created by the client in the server (for example, **Font, Pixmap, Cursor, Colormap**, and **GContext**).

Frees all colors and colormap entries allocated by a clien application.

Additional processing occurs when the last connection to the X Server closes. An X Server goes through a cycle of having no connections and having some connections. When the last display connection to the X Server closes as a result of a connection closing with the **DestroyAll** *close_mode* argument, the X Server:

Resets its state, as if it had just been started. The X Server begin by destroying all lingering resources from clients that were

¦ Copyright IBM Corp. 1985, 1991

terminated in **RetainPermanent** or **RetainTemporary** mode.

Deletes all but the predefined atom identifiers. Se "Using Properties and Atoms" in topic 1.6.4 for information about atoms.

Deletes all properties on all root windows. Se "Using Properties and Atoms" in topic 1.6.4 for information about properties.

Resets all device maps and attributes (for example, key click, bel volume, and acceleration) and the access control list.

Restores the standard root tiles and cursors

Restores the default font path

Restores the input focus to state **PointerRoot**.

However, the X Server does not reset if a connection with a *close_down mode* argument is set to **RetainPermanent** or **RetainTemporary**.

Subtopics
1.4.2.1 Related Functions

¦ Copyright IBM Corp. 1985, 1991

1.4.2.1 Related Functions

XCloseDispla

¦ Copyright IBM Corp. 1985, 1991

1.5 Using Window Functions

In X-Windows, a ***window*** is a rectangular area on the screen that lets you view graphical output. Client applications can display overlapping and nested windows on one or more screens that are driven by X Servers on one or more systems. Use **XOpenDisplay** to open a display. See "Opening a Display" in topic 1.4.1 for information about opening a display.

Subtopics
1.5.1 Defining Visual Types
1.5.2 Determining the Appropriate Visual
1.5.3 Defining Window Attributes
1.5.4 Creating Windows
1.5.5 Destroying Windows
1.5.6 Mapping Windows
1.5.7 Configuring Windows

¦ Copyright IBM Corp. 1985, 1991

1.5.1 Defining Visual Types

On some high-end displays, color resources can be used in several ways. For example, you can use the display as a 12-bit display with arbitrary mapping of pixel to color (pseudo-color) or as a 24-bit display with 8 bits of the pixel dedicated for red, green, and blue. These different ways of using visual aspects are called ***Visuals***. For example,

The screen can be color or grayscale

The screen can have a colormap that is writable or read-only

A screen can also have a colormap whose indices are decomposed int separate RGB pieces, provided one is not on a grayscale screen. The following diagram shows the color or grayscale of a colormap:

	Color		Gray Scale	
	R/O	R/W	R/O	R/W
Undecomposed Colormap	Static Color	Pseudo Color	Static Gray	Gray Scale
Decomposed Colormap	True Color	Direct Color		

¦ Copyright IBM Corp. 1985, 1991

1.5.2 Determining the Appropriate Visual

For each screen, a list of valid visual types can be supported at different depths of the display. **Xlib** uses the **XVisualInfo** data structure to determine which visual to use in the application. The **XVisualInfo** data structure is described below:

```
typedef struct {

    Visual *visual;
    VisualID visualid;
    int screen;
    unsigned int depth;
    int class;
    unsigned long red_mask;
    unsigned long green_mask;
    unsigned long blue_mask;
    int colormap_size;
    int bits_per_rgb;

} XVisualInfo;
```

Some of the members of the **XVisualInfo** data structure are *class, red_mask, green_mask, blue_mask, bits_per_rgb,* and *colormap_size*.

The *class* member specifies the possible visual classes of the screen. It can be one of the following: **PseudoColor**, **GrayScale**, **DirectColor**, **TrueColor**, **StaticColor**, or **StaticGray**.

Conceptually, as each pixel is read out of memory, it goes through a lookup stage by indexing into a colormap. Colormaps can be manipulated arbitrarily on some hardware, in a limited way on other hardware, and not at all on yet other hardware.

The visual types affect the colormap and the RGB values in the following ways:

- For **PseudoColor**, a pixel value indexes a colormap to produce independent RGB values, and the RGB values can be changed dynamically.

- For **GrayScale**, a pixel value indexes a colormap to produce independent RGB values, and the RGB values can be changed dynamically, except that the primary that drives the screen is not defined. Therefore, the client should always store the same value for red, green, and blue in the colormaps.

- For **DirectColor**, a pixel value is decomposed into separate RGB subfields, and each subfield separately indexes the colormap for the corresponding value. The RGB values can be changed dynamically.

- For **TrueColor**, a pixel value is decomposed into separate RGB subfields, and each subfield separately indexes the colormap for the corresponding value. The RGB values can be changed dynamically, except that the colormap has predefined read-only RGB values. These values are server-dependent, but provide linear or near-linear ramps in each primary.

- For **StaticColor**, a pixel value indexes a colormap to produce

¦ Copyright IBM Corp. 1985, 1991

independent RGB values, and the RGB values can be changed dynamically, except that the colormap has predefined read-only server-dependent RGB values.

- For **StaticGray**, a pixel value indexes a colormap to produce independent RGB values, and the RGB values can be changed dynamically, except that the red, green, and blue values are equal for any single pixel value which results in shades of gray. **StaticGray** with a two-entry colormap can be considered monochrome.

The *red_mask*, *green_mask*, and *blue_mask* members are defined only for **DirectColor** and **TrueColor**. Each mask has one contiguous set of bits with no intersections.

The *colormap_size* member defines the number of available colormap entries in a newly created colormap. For **DirectColor** and **TrueColor**, this number is the size of an individual pixel subfield.

The *bits_per_rgb* member specifies the log base 2 of the approximate number of distinct color values (individually) of red, green, and blue. Actual RGB values are unsigned 16-bit numbers.

The definitions for *vinfo_mask* (the visual information mask) are:

```
#define   VisualNoMask              0x0
#define   VisualIDMask              0x1
#define   VisualScreenMask          0x2
#define   VisualDepthMask           0x4
#define   VisualClassMask           0x8
#define   VisualRedMaskMask         0x10
#define   VisualGreenMaskMask       0x20
#define   VisualBlueMaskMask        0x40
#define   VisualColormapSizeMask    0x80
#define   VisualBitsPerRGBMask      0x100
#define   VisualAllMask             0x1FF
```

Subtopics
1.5.2.1 Related Functions

¦ Copyright IBM Corp. 1985, 1991

1.5.2.1 Related Functions

XGetVisualInf
XMatchVisualInf

¦ Copyright IBM Corp. 1985, 1991

1.5.3 Defining Window Attributes

All windows have a border width of zero or more pixels, an optional background, an input mask, an event suppression mask, and a property list. The window border and background can be a solid color or a pattern, which is called a ***tile***. All windows, except the root window, have a parent window and are clipped by the parent window.

If a window is stacked on top of another window, the top windo obscures the lower window, not allowing it to accept input. Input events, for example, pointer motion events, are not generated for the obscured area of the lower window.

If the top window has a background, the top window obscures the lowe window, not allowing it to have output. Output attempts to the obscured area of the lower window produce no result.

Windows with a class of **InputOnly** are used to control input events in situations where full-fledged windows are not necessary. **InputOnly** windows have the following window attributes:

win_gravity
event_mask
do_not_propagate_mask
override_redirect
cursor

If other attributes are defined for an **InputOnly** window, an error is generated.

Windows have borders of a programmable width and pattern, as well as a background pattern or tile. Pixels can be used for solid colors. Refer to the window in a program by using the resource ID of type **Window**. The background and border pixmaps can be destroyed immediately after creating the window if no further explicit references are made.

The background of a window can be a solid color or a pattern. The pattern can be relative to the parent window or absolute.

If the pattern is relative to the parent window, the pattern i shifted appropriately to match the parent window.
If the pattern is absolute, it is positioned in the windo independently of the parent window.

When an application first creates a window, the window is not visible to the screen. This window is called an ***unmapped*** window. Output to an unmapped window is discarded. When a window is eventually mapped to the screen with **XMapWindow**, the X Server generates an exposure event for the window if any client has requested this event.

A window manager can override the size, border width, and style of a window specified in your program. You should be prepared to use the actual size and position of the top window, which is reported as an event when the window is first mapped. A client program should not resize itself without command input. Instead, the client program should use the space specified. If the space specified is insufficient, the client program can request a resize of the window. Only the border of the top-level windows can be changed by window managers.

The following **XSetWindowAttributes** data structure is used to determine

¦ Copyright IBM Corp. 1985, 1991

window attributes:

```
typedef struct {

  Pixmap background_pixmap;       /* background, None, or            */
                                  ParentRelative
  unsigned long                   /* background pixel                */
background_pixel;
  Pixmap border_pixmap;           /* border of the window            */
  unsigned long border_pixel;     /* border pixel value              */
  int bit_gravity;                /* one of bit gravity values       */
  int win_gravity;                /* one of the window gravity       */
                                  values
  int backing_store;              /* NotUseful, WhenMapped, Always   */
  unsigned long backing_planes;   /* planes to be preserved if       */
                                  possible
  unsigned long backing_pixel;    /* value to use in restoring       */
                                  planes
  Bool save_under;                /* should bits under be saved      */
                                  (pop-ups)
  long event_mask;                /* set of events that should be    */
                                  saved
  long do_not_propagate_mask;     /* set of events that should not   */
                                  propagate
  Bool override_redirect;         /* Boolean value for               */
                                  override_redirect
  Colormap colormap;              /* colormap to be associated with  */
                                  window
  Cursor cursor;                  /* cursor to be displayed or None  */

} XSetWindowAttributes;
```

The following masks are defined for window functions:

```
#define   CWBackPixmap          (1L>>0)
#define   CWBackPixel           (1L>>1)
#define   CWBorderPixmap        (1L>>2)
#define   CWBorderPixel         (1L>>3)
#define   CWBitGravity          (1L>>4)
#define   CWWinGravity          (1L>>5)
#define   CWBackingStore        (1L>>6)
#define   CWBackingPlanes       (1L>>7)
#define   CWBackingPixel        (1L>>8)
#define   CWOverrideRedirect    (1L>>9)
#define   CWSaveUnder           (1L>>10)
#define   CWEventMask           (1L>>11)
#define   CWDontPropagate       (1L>>12)
#define   CWColormap            (1L>>13)
#define   CWCursor              (1L>>14)
```

The members of the **XSetWindowAttributes** data structure include the following:

background_pixmap membe

The *background_pixmap* member specifies the pixmap for a window background. This pixmap can be any size. If the *background_pixmap* is set, it overrides the default *background_pixmap*. The *background_pixmap* and the window must have the same depth.

¦ Copyright IBM Corp. 1985, 1991

The *background_pixmap* can be set to a pixmap, **None**, or **ParentRelative**. The default is **None**.

- If the *background_pixmap* is set to **None**, the window has no defined background. If the parent window has a *background_pixmap* of **None**, the window also has a *background_pixmap* of **None**.

  When regions of the window are exposed and the X Server has not retained the contents of the window, the X Server automatically tiles the regions with the window background as long as the *background_pixmap* is not **None**.

  If the *background_pixmap* is **None**, the contents of the previous screen are left in place if the window and the parent window have the same depth. Otherwise, the initial contents of the exposed regions are undefined. Exposure events are then generated for the regions, even if the *background_pixmap* is **None**. See "Processing Exposure Events" in topic 1.11.14 for a discussion of exposure event processing.

- If the *background_pixmap* is set to **ParentRelative**, the following occurs:

  -- The *background_pixmap* of the parent window is used, if the child window has the same depth as the parent window.
  -- A copy of the *background_pixmap* of the parent window is not made. The *background_pixmap* of the parent window is examined each time the *background_pixmap* of the child window is required.
  -- The background tile origin always aligns with the background tile origin of the parent window. Otherwise, the background tile origin is always the child window origin.

Setting a new background with *background_pixmap* overrides any previous border. The *background_pixmap* can be freed immediately if no further explicit reference is made to it. The X Server keeps a copy to use when needed.

background_pixel membe

The *background_pixel* member specifies a pixel value of a single color for the background of the window. This member can be set to any pixel value. The default value for *background_pixel* is undefined.

If *background_pixel* is specified, it overrides the default *background_pixmap* or any value set in the *background_pixmap* member. All pixels, in the background of the window, will be set to this value.

Setting a new background with *background_pixel* overrides any previous border.

border_pixmap membe

The *border_pixmap* member specifies the pixmap for the border of a window. This pixmap can be any size. The *border_pixmap* and the child window must have the same depth.

Setting a new border with *border_pixmap* overrides any previous border.

¦ Copyright IBM Corp. 1985, 1991

Setting a *border_pixmap* value overrides the default value. The default value is **CopyFromParent**.

If the *border_pixmap* is **CopyFromParent**, the *border_pixmap* is copied from the parent window. Subsequent changes to the border attribute of the parent window do not affect the child window.

The *border_pixmap* can be freed immediately if no further explicit reference to it is made. If the *border_pixmap* is freed, X-Windows may or may not keep a copy of it. X-Windows can use the same pixmap each time the window is repainted or it may use the option in this variable.

border_pixel membe

The *border_pixel* member specifies a pixmap of an undefined size for the window border. The border tile origin is always the same as the background tile origin. The default for *border_pixel* is undefined.

If you specify a *border_pixel*, it overrides the default value or the assigned value of *border_pixmap*. Then, all pixels in the border of the window are set to the *border_pixel* value.

Setting a new border with *border_pixel* overrides any previous border.

bit_gravity and win_gravity member

Bit gravity defines which region of the window should be retained when a window is resized. Changing the inside width or height of the window causes the contents of the window to be moved or lost depending on the *bit_gravity* of the window. The default for *bit_gravity* is **ForgetGravity**.

Window gravity defines how the window should be repositioned if the parent window is resized. Changing the inside width or height of the window causes children to be reconfigured, depending on the specified *win_gravity*. The default for *win_gravity* is **NorthWestGravity**.

If the inside width or height of a window is not changed and if the window is moved or its border is changed, then the contents of the window are not lost but are moved with the window. For a change of *width* and *height*, the (*x*, *y*) pairs are defined as follows:

Gravity Coordinates	**Direction**
NorthWestGravity	(0, 0)
NorthGravity	(Width/2, 0)
NorthEastGravity	(Width, 0)
WestGravity	(0, Height/2)
CenterGravity	(Width/2, Height/2)
EastGravity	(Width, Height/2)
SouthWestGravity	(0, Height)
SouthGravity	(Width/2, Height)

¦ Copyright IBM Corp. 1985, 1991

SouthEastGravity (Width, Height)

When a window with one of these bit_gravities is resized, the corresponding pair defines the change in position of each pixel in the window. When a window with one of these win_gravities has its parent window resized, the corresponding pair defines the change in position of the window within the parent. When the window is repositioned, a **GravityNotify** event is generated.

A *bit_gravity* of **StaticGravity** indicates that the contents or origin of the window should not move relative to the origin of the root window. If the change in size of the window is coupled with a change in position (*x*, *y*), then the following occurs:

- For *bit_gravity*, the change in position of each pixel becomes (*-x*, *-y*).
- For *win_gravity*, the change in position of a child when the parent is resized becomes (*-x*, *-y*).

StaticGravity takes effect only when the *width* or *height* of the window is changed, not when the window is moved.

A *bit_gravity* of **ForgetGravity** indicates that the contents of the window are always discarded after a size change, even if a *backing_store* or *save_under* has been requested. The window is tiled with its background, and one or more exposure events are generated. If no background is defined, the existing screen contents are not altered. Some X Servers may ignore the specified *bit_gravity* and always generate exposure events.

A *win_gravity* of **UnmapGravity** is like **NorthWestGravity**; the window is not moved, but the child is unmapped when the parent is resized, and an **UnmapNotify** event is generated.

backing_store membe

The *backing_store* member advises the X Server what to do with the contents of a window. ***backing-store*** is the pixels saved offscreen. This member can be set to **NotUseful**, **WhenMapped**, or **Always**. The default is **NotUseful**.

- **NotUseful** advises the X Server that maintaining contents is not necessary. Some X-Windows implementations can still maintain contents; however, exposure events are not generated.

- **WhenMapped** advises the X Server that maintaining contents of obscured regions when the window is mapped would be beneficial.

- **Always** advises the X Server that maintaining contents even when the window is unmapped would be beneficial. Even if the window is larger than the parent window, this requests that the X Server maintain the complete contents of the window, not just the contents of the region within the boundaries of the parent window. If the X Server maintains the contents of the window, exposure events are not generated. The X Server can stop maintaining contents at any time.

backing_planes membe

¦ Copyright IBM Corp. 1985, 1991

The *backing_planes* member indicates (with one bit) which bit planes of the window hold dynamic data that must be preserved in *backing_store* and during *save_under*. The default for *backing_planes* is 1.

backing_pixel membe

The *backing_pixel* member specifies the values to use in planes not covered by *backing_planes*. The X Server is free to save only the specified bit planes in the *backing_store*. Any extraneous bits in these values can be ignored. The default for *backing_pixel* is zero.

save_under membe

The *save_under* member can regenerate the remaining planes with the specified pixel value. If *save_under* is **True**, the X Server is advised that saving the contents of the windows that it obscures would be beneficial when this window is mapped. The default for *save_under* is **False**.

Some server implementations can preserve bits of windows under other windows. This is not the same as preserving the contents of a window. If transient windows, such as pop-up menus, request that the system preserve the bits under them, the temporarily obscured applications do not have to repaint.

event_mask and do_not_propagate_mask member

The *event_mask* member defines events the client is interested in for this window or, in some cases, for the inferiors of the window.

The *do_not_propagate_mask* member defines events that should not be propagated to ancestor windows when no client has the event type selected in this window. These masks are the bitwise inclusive-OR gates of one or more of the valid event mask bits. If the constant **NoEventMask** is specified, no maskable events are reported.

The default for *event_mask* and *do_not_propagate_mask* is the empty set. See "Defining Event Masks" in topic 1.11.3 for information on event masks and events.

override_redirect membe

The *override_redirect* member specifies whether or not a map or configure request on a window should override a **SubstructureRedirectMask** on the parent window. The default is **False**.

To control window placement or to add decoration, a window manager may need to intercept or redirect a map or configure request. Pop-up windows, however, need to be mapped so that a window manager does not interfere with the response they receive. To do this, *override_redirect* must be used.

colormap membe

The *colormap* member specifies the colormap, if any, that best reflects the true colors of the window. The *colormap* is copied by sharing the colormap object between the child and the parent, not by making a complete copy of the colormap contents. The colormap must have the same visual type as the window. The parent window must not have a

¦ Copyright IBM Corp. 1985, 1991

colormap of **None**.

If the *colormap* member is set to **CopyFromParent**, the colormap of the parent window is copied and used by the child window. Subsequent changes to the parent window do not affect the child window. However, the child window must have the same visual type as the parent.

The default for *colormap* is **CopyFromParent**.

cursor membe

If a *cursor* is specified, it is used whenever the pointer is in the specified window. The default is **None**.

If **None** is specified, the parent cursor is used when the pointer is in the window. Any change in the parent cursor will change the displayed cursor immediately. Use **XFreeCursor** to free the cursor immediately if no further explicit reference to it is made. See "Manipulating Cursors" in topic 1.9.4 for further information.

¦ Copyright IBM Corp. 1985, 1991

1.5.4 Creating Windows

Xlib provides basic ways for creating windows. When you create top-level windows or direct children of the root window, the following should be observed to ensure that applications interact properly across differing styles of window management.

Allow the window manager to determine the size or placement of you top-level windows. Provide the window manager with some standard information or hints with the various window manager hints functions. See "Setting and Getting Window Manager Hints" in topic 1.12.4.

An application, by interpreting the first exposure event, must be abl to deal with whatever size window it gets, even if this means that the application just prints a message, such as "Please make me bigger", in the window.

An application should be able to resize or move the children of it top-level window as necessary. An application should only resize or move its top-level window in direct response to a user request. Otherwise, the request may fail.

Applications should not be written to assume control of the windo manager.

The application should set standard window properties for th top-level window before mapping it. To set standard window properties for a top-level window, use **XSetStandardProperties**. See "Using Predefined Property Functions" in topic 1.12 for more information.

Use **XCreateWindow** and **XCreateSimpleWindow** to create an unmapped subwindow for a specified parent window.

XCreateWindow allows you to set specific window attributes when you create the window.

XCreateSimpleWindow creates a window that inherits its window attributes from the parent window.

InputOnly windows cannot be used for graphics requests, exposure processing, and **VisibilityNotify** events. An **InputOnly** window cannot be used as a source or destination drawable for graphics requests.

Subtopics
1.5.4.1 Related Functions

¦ Copyright IBM Corp. 1985, 1991

1.5.4.1 Related Functions

XCreateWindo

XCreateSimpleWindo

¦ Copyright IBM Corp. 1985, 1991

1.5.5 Destroying Windows

X-Windows also allows you to destroy windows and subwindows. You can destroy a window or destroy all subwindows of a window. If a window is destroyed, it should not be referenced by another function. If you specify that a root window be destroyed, no windows are actually destroyed. By default, windows are destroyed when a connection to the X Server is closed.

Subtopics
1.5.5.1 Related Functions

¦ Copyright IBM Corp. 1985, 1991

1.5.5.1 Related Functions

XDestroyWindo

XDestroySubwindo

¦ Copyright IBM Corp. 1985, 1991

1.5.6 Mapping Windows

A window manager can control the placement of subwindows. If **SubstructureRedirectMask** has been selected by a window manager on a parent window (usually a root window), a map request initiated by other clients on a child window is not performed, and the window manager is sent a **MapRequest** event.

If the *override_redirect* flag on the child is set to **True** (usually only on pop-up menus), the map request is performed.

A tiling window manager can reposition and resize other client windows and then map the window at its final location. Only one client at a time can select **SubstructureRedirectMask**.

Similarly, a single client can select **ResizeRedirectMask** on a parent window. Any attempt to resize the window is suppressed, and the client, which is usually a window manager, receives a **ResizeRequest** event. These mechanisms allow arbitrary placement policy to be enforced by an external window manager.

A window is considered mapped if an **XMapWindow** call is made on the window. The window may not be visible on the screen for one of the following reasons:

The window is hidden by another opaque sibling window
One of the window ancestors is not mapped
The window is entirely clipped by an ancestor

Exposure events are generated for the window when part or all of it becomes visible on the screen. A client will only receive the exposure events if it requests these events with **XSelectInput**. Windows retain their position in the stacking order when unmapped.

Subtopics
1.5.6.1 Related Functions

¦ Copyright IBM Corp. 1985, 1991

1.5.6.1 Related Functions

XMapWindow

XMapSubwindows

XUnmapSubwindows

XMapRaised

XUnmapWindow

¦ Copyright IBM Corp. 1985, 1991

1.5.7 Configuring Windows

Xlib functions move a window, resize a window, move and resize a window, or change the border width of a window. The most general interface to configuring windows, **XConfigureWindow**, uses the **XWindowChanges** data structure, which contains the following defined masks:

```
#define   CWX                 (1L>>0)
#define   CWY                 (1L>>1)
#define   CWWidth             (1L>>2)
#define   CWHeight            (1L>>3)
#define   CWBorderWidth       (1L>>4)
#define   CWSibling           (1L>>5)
#define   CWStackMode         (1L>>6)
```

The **XWindowChanges** data structure is described below:

```
typedef struct {

     int x, y;
     int width, height;
     int border_width;
     Window sibling;
     int stack_mode;

     } XWindowChanges
```

The members of the **XWindowChanges** data structure include the following:

The *x* and *y* members which indicate the position of the upper-left corner of the window. These coordinates are relative to the origin of the parent window.

The *width* and *height* which specify the size of the inside of the window, excluding the border. These arguments should be non-zero. Attempts to configure a root window have no effect.

The *border_width* which specifies the width of the border in pixels. Changing only the *border_width* leaves the outer-left corner of the window in a fixed position, but moves the absolute position of the window origin.

The *sibling* which specifies the sibling window for stacking operations.

The *stack_mode* member which can be **Above**, **Below**, **TopIf**, **BottomIf**, or **Opposite**.

If a *sibling* and a *stack_mode* are specified, the window is restacked as follows:

stack_mode	**sibling specified**
Above	The window is placed just above the sibling.
Below	The window is placed just below the sibling.
TopIf	If the sibling occludes the window, the window is placed at the top of the stack.

¦ Copyright IBM Corp. 1985, 1991

BottomIf If the window occludes the sibling, the window is placed at the bottom of the stack.

Opposite If the sibling occludes the window, the window is placed at the top of the stack. Otherwise, if the window occludes the sibling, the window is placed at the bottom of the stack.

If a *stack_mode* is specified without a sibling, the window is restacked as follows:

stack_mode	**sibling not specified**
Above	The window is placed at the top of the stack.
Below	The window is placed at the bottom of the stack.
TopIf	If any sibling occludes the window, the window is placed at the top of the stack.
BottomIf	If the window occludes any sibling, the window is placed at the bottom of the stack.
Opposite	If any sibling occludes the window, the window is placed at the top of the stack. Otherwise, if the window occludes any sibling, the window is placed at the bottom of the stack.

If the *override_redirect* attribute of the window is **False**, and if some other client has selected **SubstructureRedirectMask** on the parent, then the X Server generates a **ConfigureRequest** event, and no further processing is performed.

Otherwise, if some other client has selected **ResizeRedirectMask** on the window and the inside width or height of the window is being changed, a **ResizeRequest** event is generated, and the current inside width and height are used instead.

The *override_redirect* attribute of the window does not affect **ResizeRedirectMask**, and **SubstructureRedirectMask** on the parent has precedence over **ResizeRedirectMask** on the window.

When the geometry of the window is changed as specified, the window is restacked among siblings, and a **ConfigureNotify** event is generated. X-Windows generates **GravityNotify** events after generating **ConfigureNotify** events. If the inside width or height of the window has changed, then children of the window are affected as follows:

If the size of the window actually changes, the subwindow may mov according to the window gravity. Depending on the window's bit gravity, the contents of the window may also be moved. See "Defining Window Attributes" in topic 1.5.3.

If regions of the window were obscured, but are not now, exposur processing is performed on formerly obscured windows, including the window itself and its inferiors.

By increasing the width or height, exposure processing is als performed on any new regions of the window and on any regions where window contents are lost.

¦ Copyright IBM Corp. 1985, 1991

The restack check, specifically the computation for **BottomIf**, **TopIf**, and **Opposite**, is performed with respect to the final size and position of the window as controlled by the other arguments to the request, not the initial position of the window. A *sibling* member should be specified with a *stack_mode* member.

Subtopics
1.5.7.1 Related Functions

¦ Copyright IBM Corp. 1985, 1991

1.5.7.1 Related Functions

XConfigureWindow

XMoveWindow

XResizeWindow

XMoveResizeWindow

XSetWindowBorderWidth

XRaiseWindow

XLowerWindow

XCirculateSubwindows

XCirculateSubwindowsUp

XCirculateSubwindowsDown

XRestackWindows

XChangeWindowAttributes

XSetWindowBackground

XSetWindowBackgroundPixmap

XSetWindowBorder

XSetWindowBorderPixmap

XTranslateCoordinates

¦ Copyright IBM Corp. 1985, 1991

1.6 Using Window Information Functions

After the display is connected to the X Server and a window is created, use **Xlib** window information functions to:

- Obtain information about a windo
- Manipulate property list
- Obtain and change window propertie
- Manipulate window selection

Subtopics

¦ Copyright IBM Corp. 1985, 1991

1.6.1 Defining Coordinates

Like the display screen, each window has its own XY coordinate system with the origin at the upper-left hand corner. The X axis is horizontal (left to right); the Y axis is vertical (top to bottom). Each addressable point on the screen is called a ***pixel*** and corresponds to one XY point in the coordinate system. Because each window has its own coordinate system, operations within windows can be insensitive to the window position on the screen.

The origin of a window is inside the border, if it has one, and window size is always the usable number of pixels within the border. The window border is maintained by the X Server, and output to the window is clipped so as not to extend into the border.

¦ Copyright IBM Corp. 1985, 1991

1.6.2 Obtaining Window Information

Xlib functions obtain information about the window tree, the current attributes of a window, its current geometry, or the current pointer coordinates. Because these functions are used primarily by window managers, they return a status to indicate if the window still exists.

The **XGetWindowAttributes** function returns the current attributes for the specified window to an **XWindowAttributes** structure. The data structure is defined as:

typedef struct {

int *x, y*;	/* location of window	*/
int *width, height*;	/* width and height of window	*/
int *border_width*;	/* border width of window	*/
int *depth*;	/* depth of window	*/
Visual **visual*;	/* the associated visual structure	*/
Window *root*;	/* root of screen containing window	*/
int *class*;	/* InputOutput, InputOnly	*/
int *bit_gravity*;	/* one of the bit gravity values	*/
int *win_gravity*;	/* one of the window gravity values	*/
int *backing_store*;	/* NotUseful, WhenMapped, Always	*/
unsigned long *backing_planes*;	/* planes to be preserved if possible	*/
unsigned long *backing_pixel*;	/* value to be used when restoring planes	*/
Bool *save_under*;	/* Boolean, should bits under be saved	*/
Colormap *colormap*;	/* colormap to be associated with window	*/
Bool *map_installed*;	/* Boolean, is colormap currently installed	*/
int *map_state*;	/* IsUnmapped, IsUnviewable, IsViewable	*/
long *all_event_masks*;	/* set of events all people have interest in	*/
long *your_event_mask*;	/* my event mask	*/
long	/* set of events that should not	*/

Copyright IBM Corp. 1985, 1991

```
¦ do_not_propagate_mask;       ¦ propagate                              ¦    ¦
+------------------------------+----------------------------------------+----¦
¦    Bool override_redirect;   ¦ /* boolean value for                    ¦ */ ¦
¦                              ¦ override-redirect                      ¦    ¦
+------------------------------+----------------------------------------+----¦
¦    Screen *screen;           ¦ /* back pointer to correct screen       ¦ */ ¦
+-------------------------------------------------------------------------+
```

} **XWindowAttributes**;

The members of the **XWindowAttributes** data structure are:

The *x* and *y* members which define the location of the drawable. If the drawable is a window, these coordinates specify the upper-left outer-corner of the window relative to the origin of the parent window. If the drawable is a pixmap, these coordinates are set to zero.

The *width* and *height* which specify the size of the inside of the window, excluding the border, for a window.

The *border_width* which specifies the width of the border in pixels. If the drawable is a pixmap, this variable is set to zero.

The *depth* which is set to the depth of the pixmap, bits per pixel for the object. The depth must be supported by the root of the specified drawable.

The *visual* member which is a pointer to the **Visual** structure associated with the screen. See "Determining the Appropriate Visual" in topic 1.5.2.

The *root* member which is set to the root ID of the screen containing the window.

The *class* which is set to the window class of **InputOutput** or **InputOnly**.

The *bit_gravity* which is set to the bit gravity of the window. See page 1.5.3 for the values.

The *win_gravity* member which is set to the gravity of the window. See page 1.5.3 for the values.

The *backing_store* member which indicates how the X Server should maintain the contents of a window. It can be set to **NotUseful**, **WhenMapped**, or **Always**. See page 1.5.3 for more information.

The *backing_planes* member which indicates which bit planes of the window that hold dynamic data must be preserved in *backing_store* and during *save_under*. See page 1.5.3 for more information.

The *backing_pixel* member which indicates the values to use when restoring planes from a partial backing store. See page 1.5.3 for more information.

The *save_under* member which indicates if the area under the newly mapped windows should be saved and restored. It can be **True** or **False**. See page 1.5.3 for more information.

¦ Copyright IBM Corp. 1985, 1991

The *colormap* member which is set to the colormap for the window specified. It can be set to a colormap ID or to **None**.

The *map_installed* which indicates if the colormap is currently installed. It can be **True** or **False**.

The *map_state* which indicates the state of the window. It can be **IsUnmapped**, **IsUnviewable**, or **IsViewable**. If the window is mapped, but an ancestor is unmapped, *map_state* is set to **IsUnviewable**.

The *all_event_masks* member which is set to the bitwise inclusive-OR of all event masks selected on the window by interested clients. See "Defining Event Masks" in topic 1.11.3 for a discussion of events and event masks.

The *your_event_mask* member which is set to the bitwise inclusive-OR of all event masks selected by the querying client. See "Defining Event Masks" in topic 1.11.3 for a discussion of events and event masks.

The *do_not_propagate_mask* member which is set to the bitwise inclusive-OR gate or operation of the set of events that should not propagate. See "Defining Event Masks" in topic 1.11.3 for a discussion of events and event masks.

The *override_redirect* member which indicates if this window overrides structure control facilities. It can be **True** or **False**. If *override_redirect* is **True**, window manager clients usually should ignore the window.

Transient windows should mark the windows associated with them. See "Setting and Getting the Transient Property" in topic 1.12.8 for further information.

The *screen* member which returns a pointer to the correct screen.

See "Defining Window Attributes" in topic 1.5.3 for more information.

Subtopics
1.6.2.1 Related Functions

¦ Copyright IBM Corp. 1985, 1991

1.6.2.1 Related Functions

XQueryTre

XGetWindowAttribute

XGetGeometr

XQueryPointe

¦ Copyright IBM Corp. 1985, 1991

1.6.3 Obtaining the X-Windows Environment Defaults

A program often needs a variety of options in the X environment (for example, fonts, colors, mouse, background, text, and cursor). Specifying these options to run on the command line is inefficient and unmanageable because individual users have different tastes with regard to window appearance. On the other hand, **XGetDefault** facilitates finding the default fonts, colors, and other environment options favored by a particular user.

Defaults are usually loaded into the resource manager property on the root window during login. **XGetDefault** merges additional defaults from a file in the user's home directory. **XGetDefault** provides a simple interface for clients.

Subtopics
1.6.3.1 Related Functions

¦ Copyright IBM Corp. 1985, 1991

1.6.3.1 Related Functions

XGetDefaul

¦ Copyright IBM Corp. 1985, 1991

1.6.4 Using Properties and Atoms

A ***property*** is a collection of named typed data. Data of only one type may be associated with a single property. Each property has a name. A unique identifier or ***atom*** is associated with each named property. An atom is a nickname for a property and is used so that the protocol does not have to send arbitrary length property names back and forth. For efficiency reasons, atoms are used instead of the whole name string. Clients store and retrieve properties associated with windows.

Windows have a set of predefined properties, for example, the name of a window, size hints, and other properties. Users can define any other arbitrary information and can associate this information with a window. (Use **XInternAtom** to define new properties or to obtain the atom for new properties.)

A property also has a type, for example, a string or an integer. These types are also indicated using atoms. Therefore, arbitrary new types can be defined. The type of a property is defined by other properties, which allows for arbitrary extension. Use the properties mechanism to communicate other information between applications.

A property is stored in one of several possible formats. The X Server can store the information as 8-bit, 16-bit, or 32-bit quantities. This flexibility permits the X Server to present the data in the byte order that the client expects.

Certain properties are predefined in the server for commonly used functions. The atoms for these properties are defined in **<X11/Xatom.h>**. To avoid name clashes with user symbols, the macro name for each atom has the **XA_** prefix added. See "Obtaining and Changing Window Properties" in topic 1.6.4.2. See "Using Predefined Property Functions" in topic 1.12 for an explanation of the functions that get and set the information stored in the predefined properties.

Atoms occur in five distinct name spaces within the protocol: selections, property names, property types, font properties and types of **ClientMessage** events (none are built into the X Server).

Any particular atom can have some client interpretation within each of the name spaces. The built-in selection properties, which name properties, are **PRIMARY** and **SECONDARY**.

The built-in property names are:

CUT_BUFFER0	RGB_GREEN_MAP	WM_CLIENT_MACHINE
CUT_BUFFER1	RGB_RED_MAP	WM_COMMAND
CUT_BUFFER2	RGB_BEST_MAP	WM_HINTS
CUT_BUFFER3	RGB_BLUE_MAP	WM_ICON_NAME
CUT_BUFFER4	RGB_DEFAULT_MAP	WM_ICON_SIZE
CUT_BUFFER5	RGB_GRAY_MAP	WM_NAME
CUT_BUFFER6	RESOURCE_MANAGER	WM_NORMAL_HINTS
CUT_BUFFER7	WM_CLASS	WM_ZOOM_HINTS
	WM_TRANSIENT_FOR	

The built-in property types are:

ARC	DRAWABLE	RGB_COLOR_MAP
ATOM	FONT	STRING
BITMAP	INTEGER	VISUALID

¦ Copyright IBM Corp. 1985, 1991

CARDINAL	PIXMAP	WINDOW
COLORMAP	POINT	WM_HINTS
CURSOR	RECTANGLE	WM_SIZE_HINTS

The built-in font property types are:

MIN_SPACE	UNDERLINE_POSITION	QUAD_WIDTH
NORM_SPACE	UNDERLINE_THICKNESS	WEIGHT
MAX_SPACE	FONT_NAME	POINT_SIZE
END_SPACE	FULL_NAME	RESOLUTION
SUPERSCRIPT_X	STRIKEOUT_DESCENT	COPYRIGHT
SUPERSCRIPT_Y	STRIKEOUT_ASCENT	NOTICE
SUBSCRIPT_X	ITALIC_ANGLE	FAMILY_NAME
SUBSCRIPT_Y	X_HEIGHT	CAP_HEIGHT

Subtopics
1.6.4.1 Related Functions
1.6.4.2 Obtaining and Changing Window Properties
1.6.4.3 Related Functions

¦ Copyright IBM Corp. 1985, 1991

1.6.4.1 Related Functions

XInternAto

XGetAtomNam

¦ Copyright IBM Corp. 1985, 1991

1.6.4.2 Obtaining and Changing Window Properties

You can attach a property list to every window. Each property has a name, a type, and a value. The value is an array of 8-bit, 16-bit, or 32-bit quantities, whose interpretation is left to the clients.

You can obtain, rotate, or change a window property. In addition, **Xlib** provides functions for predefined property operations. See "Using Predefined Property Functions" in topic 1.12.

¦ Copyright IBM Corp. 1985, 1991

1.6.4.3 Related Functions

XGetWindowPropert

XListPropertie

XChangePropert

XRotateWindowPropertie

XDeletePropert

¦ Copyright IBM Corp. 1985, 1991

1.6.5 Using Window Selections

Selection is one method of exchanging data between applications. By using the property mechanism, applications can exchange data of arbitrary types and can negotiate the type of data to be exchanged. A selection can be thought of as an indirect property with a dynamic type. Rather than having the property stored in the X Server, the property is maintained by some client (the owner). A selection is global in nature. It belongs to the user but is maintained by the clients, rather than being private to a particular window subhierarchy or a particular set of clients.

Xlib functions set, get, or convert window selection. These functions allow applications to implement the notion of current selection, which requires that applications be notified when they no longer own the selection. Applications that support selection often highlight the current selection. To unhighlight the selection, applications must be able to be informed when some other application acquires the selection.

When a client asks for the contents of a selection, it specifies a selection target type. This target type can be used to control the transmitted representation of the contents. For example, if the selection is "the last thing the user clicked on", and currently an image, then the target type might specify whether the contents of the image should be sent in **XYFormat** or **ZFormat**.

The target type can also be used to control the class of contents transmitted, for example, asking for the page format (fonts, line spacing, indentation, and other page format specifications) of a paragraph selection, not the text of the paragraph. The target type can also be used for other purposes. The semantics are not constrained by the protocol.

Subtopics
1.6.5.1 Related Functions

¦ Copyright IBM Corp. 1985, 1991

1.6.5.1 Related Functions

XSetSelectionOwne

XGetSelectionOwne

XConvertSelectio

¦ Copyright IBM Corp. 1985, 1991

1.7 Using Graphics Resource Functions

X-Windows graphics resource functions allow you to manipulate colormaps, pixmaps, graphics context (GC) state and use GC convenience routines.

A number of resources are used when performing graphics operations in X-Windows. Most attributes of graphics operations (for example, foreground color, background color, line style, and other graphics) are stored in resources called ***graphics contexts*** (GC). Most graphics operations take a graphics context (GC) as an argument. While sharing GCs between applications is possible, it is not encouraged. Applications should use their own GCs when performing operations.

Windows have a colormap associated with them that provides a level of indirection between pixel values and color displayed on the screen. Many hardware displays have a single colormap; therefore, the primitives are written to share colormap entries between applications. Because colormaps are associated with windows, X-Windows supports displays with multiple colormaps and different types of colormaps. If there are not enough colormap resources in the display, some windows may not be displayed in their true colors. A window manager can set the displayed windows in their true colors if more than one colormap is required for the color resources the applications are using.

Off-screen memory or pixmaps are often used to define frequently-used images for later use in graphics operations. Pixmaps are also used to define tiles or patterns for window backgrounds, borders, or cursors. (A single-bit plane pixmap is sometimes referred to as a ***bitmap***. See "Defining Bitmaps" in topic 1.7.7.1.) There may not be an unlimited amount of off-screen memory; therefore, it should be regarded as a precious resource.

Graphics operations can be performed to windows or pixmaps, which are also called drawables. Each drawable exists on a single screen or root. The drawable can be used only on that root. GCs can be used with drawables of matching roots and depths.

Subtopics

¦ Copyright IBM Corp. 1985, 1991

1.7.1 Creating Colormaps

Xlib provides functions to create, allocate, free, set and manipulate colormaps. The introduction of color changes the view a programmer should take when dealing with a bitmap display. For example, when printing text, you write in a color or pixel value rather than by setting or clearing bits. Hardware imposes limits (number of significant bits, for example). Typically, you allocate particular pixel values or sets of values. If read-only, the pixel values can be shared among multiple applications. If read/write, the pixel values are exclusively owned by the program, and the color cell associated with the pixel value can be changed at will.

Each pixel on a monochrome (black and white) display has one bit of information associated with it; the bit is either zero (0) or one (1).

Color displays, however, need multiple bits per pixel to properly regulate the red, green, and blue color guns that provide the full range of visible colors available to the particular display. For example, assume a display has 4 bits per pixel, numbered 0, 1, 2, and 3. The display is said to be 4 planes deep. One plane contains all the bit 0s, the second contains the bit 1s, the third contains the bit 2s, and the fourth contains the bit 3s.

Some functions manipulate the representation of color on the screen. For each possible value a pixel can take on a display, a color cell is defined in the colormap. For example, if a display is 4 bits deep, pixel values 0 through 15 are defined. A ***colormap*** is a collection of the color cells. A color cell consists of a triple of red, green, and blue. As each pixel is read out of display memory, its value is taken and looked up in the colormap. The values of the cell determine what color is displayed on the screen. On a multiplane display with a black and white monitor (grayscale, but not color), these values may be combined to determine the brightness on the screen. See "Manipulating Standard Colormaps" in topic 1.7.3.

One or more colormaps may reside at one time on certain hardware. **XInstallColormap** installs a colormap; while **DefaultColormap** returns the default colormap. The **DefaultVisual** macro returns the default visual type for the specified screen. Colormaps are local to a particular screen. Possible visual types are represented by **StaticGray**, **GrayScale**, **StaticColor**, **PseudoColor**, **TrueColor**, or **DirectColor**. See "Determining the Appropriate Visual" in topic 1.5.2.

The following is the data structure for **XColor**:

typedef struct {

unsigned long *pixel*;	/* pixel value	*/
unsigned short *red, green, blue*;	/* rgb values	*/
char *flags*;	/* DoRed, DoGreen, DoBlue	*/
char *pad*;		

} **XColor**;

The *red*, *green*, and *blue* values are scaled between 0 and 65535. That is,

¦ Copyright IBM Corp. 1985, 1991

on full, the color value would be 65535 independent of the number of bit planes of the display. For half brightness in a color, the value would be 32767. The color value would be 0 for off. This value gives uniform results for color values across displays with different numbers of bit planes.

The *flags* member, which can be one or more of **DoRed**, **DoGreen**, or **DoBlue**, is used in some functions to control which members are set.

Subtopics
1.7.1.1 Related Functions

¦ Copyright IBM Corp. 1985, 1991

1.7.1.1 Related Functions

XCreateColorma

XCopyColormapAndFre

XSetWindowColorma

XFreeColorma

¦ Copyright IBM Corp. 1985, 1991

1.7.2 Allocating Colormaps

Xlib provides functions to allocate or deallocate pixel values for colors that you need to display. There are two ways of allocating color cells: explicitly as **read only** entries by pixel value or as **read/write** entries, where you can allocate a number of color cells and planes simultaneously. The read or write cells allocated are not defined colors until the colors are set with **XStoreColor** or **XStoreColors**.

Screens always have a default colormap. Programs typically allocate cells out of a common colormap. Writing applications that monopolize color resources is discouraged. On a screen that cannot load the colormap or cannot have a fully independent colormap, only certain kinds of allocations work.

To determine the color names, the X Server uses a color database which is stored in **/usr/lpp/X11/rgb/rgb**. A printable copy of this database is stored in **/usr/lpp/X11/rgb/rgb.txt**.

The name and contents of this file are operating-system specific and possibly screen specific. Although you can change the values in a read/write color cell that is allocated by another application, this is not encouraged. See "Creating Colormaps" in topic 1.7.1 for information about the **XColor** data structure.

Subtopics
1.7.2.1 Related Functions

¦ Copyright IBM Corp. 1985, 1991

1.7.2.1 Related Functions

XAllocColor

XAllocNamedColor

XLookupColor

XAllocColorCells

XAllocColorPlanes

XStoreColors

XStoreColor

XStoreNamedColor

XFreeColors

¦ Copyright IBM Corp. 1985, 1991

1.7.3 Manipulating Standard Colormaps

Applications with color palettes, smooth-shaded drawings, or digitized images demand large numbers of colors. In addition, these applications often require an efficient mapping from color triples to pixel values that display the appropriate colors.

As an example, consider a 3D display program designed to draw a smoothly shaded sphere. At each pixel in the image of the sphere, the program computes the intensity and color of light reflected back to the viewer. The result of each computation is a triple of red, green, and blue co-efficients in the range 0.0 to 1.0. To draw the sphere, the program needs a colormap that provides a large range of uniformly distributed colors. The colormap should be arranged so that the program can convert its RGB triples into pixel values very quickly because drawing the entire sphere requires many such conversions.

On many current workstations the display is limited to 256 or fewer colors. Applications must allocate colors carefully, not only to make sure they cover the entire range needed, but also to make use of as many of the available colors as possible. On a typical X Server display, many applications are active at once. Most workstations have only one hardware lookup table for colors; therefore, only one application colormap can be installed at a given time. The application using the installed colormap is displayed correctly while the other applications are displayed with false colors.

As another example, consider users who run an image processing program to display earth-resources data. The image processing program needs a colormap set up with 8 reds, 8 greens, and 4 blues which is a total of 256 colors. Because some colors are already in use in the default colormap, the image processing program allocates and installs a new colormap.

The users decide to alter some of the colors in the image. They invoke a color palette program to mix and choose colors. The color palette program also needs a colormap with 8 reds, 8 greens, and 4 blues; therefore, just as the image-processing program, the color palette program must allocate and install a new colormap.

Because only one colormap can be installed at a time, the color palette can be displayed incorrectly when the image-processing program is active. Conversely, when the palette program is active, the image can be displayed incorrectly. Users can never match or compare colors in the palette and image. Contention for colormap resources can be reduced if applications with similar color needs share colormaps.

As another example, the image processing program and the color palette program share the same colormap if there exists a convention that describes how the colormap is set up. Whenever either program is active, both are displayed correctly.

The standard colormap properties define a set of commonly used colormaps. Applications that share these colormaps and conventions display true colors more often and provide a better interface to the users.

¦ Copyright IBM Corp. 1985, 1991

1.7.4 Using Standard Colormaps

Standard colormaps allow applications to share commonly used color resources. This feature allows many applications to be displayed in true colors simultaneously, even when each application needs an entirely filled colormap.

Several standard colormaps are described. Usually, these colormaps are created by a window manager. Applications should use the standard colormaps if they already exist. If the standard colormaps do not exist, applications should create new standard colormaps.

The **XStandardColormap** data structure contains:

```
typedef struct {

    Colormap colormap;
    unsigned long red_max;
    unsigned long red_mult;
    unsigned long green_max;
    unsigned long green_mult;
    unsigned long blue_max;
    unsigned long blue_mult;
    unsigned long base_pixel;

} XStandardColormap;
```

The properties containing the **XStandardColormap** data structure have the type **RGB_COLOR_MAP**.

The members in the **XStandardColormap** structure include the following:

The *colormap* which is the ID of a colormap created by the **XCreateColormap** function.

The *red_max* which gives the maximum red values. The *green_max* which gives the maximum green values. The *blue_max* which gives the maximum blue values. Each color coefficient ranges from zero (0) to its maximum, inclusive.

An example of a common colormap allocation is **3/3/2** (**3** planes for red, **3** planes for green, and **2** planes for blue). This colormap would have *red_max* = **7**, *green_max* = **7**, and *blue_max* = **3**.

An alternate allocation that uses only 216 colors is *red_max* = **5**, *green_max* = **5**, and *blue_max* = **5**.

The *red_mult*, *green_mult*, and *blue_mult* which give the scale factors used to compose a full pixel value.

For a **3/3/2** allocation *red_mult* might be **32**, *green_mult* might be **4**, and *blue_mult* might be **1**

For a 6-colors-each allocation, *red_mult* might be **36**, *green_mult* might be **6**, and *blue_mult* might be **1**.

The *base_pixel* which gives the base pixel value used to compose a full pixel value. The *base_pixel* is obtained usually from the **XAllocColorPlanes** function.

¦ Copyright IBM Corp. 1985, 1991

Given integer red, green, and blue co-efficients in the appropriate ranges, you can compute a corresponding pixel value by using the following expression:

```
r * red_mult + g * green_mult + b * blue_mult + base_pixel
```

For gray-scale colormaps, only the *colormap*, *red_max*, *red_mult*, and *base_pixel* fields are defined. The other fields are ignored. Compute a gray-scale pixel value by using the following expression:

```
gray * red_mult + base_pixel
```

¦ Copyright IBM Corp. 1985, 1991

1.7.5 Using Standard Colormap Properties and Atoms

Several standard colormaps are available. Each standard colormap is defined by a property, and each property is identified by an atom. The following is a list of atom names and the colormap associated with each one.

The **RGB_DEFAULT_MAP** atom names a property with the value of **XStandardColormap**. The property defines an RGB subset of the system default colormap. Some applications need only a few RGB colors and may be able to allocate these colors from the system default colormap. By using the system default colormap, you ensure that there are fewer active colormaps in the system. This, in turn, ensures that your application is more likely to be displayed with the correct colors at all times.

A typical allocation for the **RGB_DEFAULT_MAP** on 8-plane displays is 6 reds, 6 greens, and 6 blues. This allocation gives 216 uniformly distributed colors (6 intensities of 36 different hues) and still leaves 40 elements of a 256-element colormap available for special-purpose colors for text, borders and other fields.

The **RGB_BEST_MAP** atom names a property with the value of **XStandardColormap**. The property defines the best RGB colormap available on the display.

Many image processing and 3D applications need to use all available colormap cells and distribute as many perceptually distinct colors as possible over those colormap cells. This implies that more green values may be available than red values, as well as more green values or red values than blue values.

On an 8-plane pseudocolor display, **RGB_BEST_MAP** is a **3/3/2** allocation. On a 24-plane direct color display, **RGB_BEST_MAP** is an **8/8/8** allocation. On other displays, **RGB_BEST_MAP** allocation is set by the user or the owner of the display.

The **RGB_RED_MAP** atom names a property with the value of **XStandardColormaps**. This property defines an all-red colormap, which is used to make color-separated images.

The **RGB_GREEN_MAP** atom names a property with the value of **XStandardColormaps**. This property defines an all-green colormap, which is used to make color-separated images.

The **RGB_BLUE_MAP** atom names a property with the value of **XStandardColormaps**. This property defines an all-blue colormap, which is used to make color-separated images.

For example, a user might generate a full-color image on an 8-plane display both by rendering an image three times (once with high color resolution in red, once with high color resolution in green, and once with high color resolution in blue) and by multiple-exposing a single frame in a camera.

The **RGB_GRAY_MAP** atom names a property with the value of **XStandardColormap**. This property describes the best gray-scale colormap available on the display. See "Using Standard Colormaps" in topic 1.7.4.

¦ Copyright IBM Corp. 1985, 1991

Subtopics
1.7.5.1 Related Functions

¦ Copyright IBM Corp. 1985, 1991

1.7.5.1 Related Functions

XGetStandardColorma

XSetStandardColorma

¦ Copyright IBM Corp. 1985, 1991

1.7.6 Determining Resident Colormaps

Window manager applications usually install and uninstall colormaps. Thus, these tasks should not be performed by normal client applications.

The X Server maintains a subset of the installed colormaps in an ordered list called the ***required list***. The length of the required list is the minimum number of installed colormaps specified for the screen when the connection is opened to the server. Initially, only the default colormap for a screen is installed, but it is not in the required list. The X Server maintains the required list as follows:

> If a colormap resource ID is passed to the *cmap* argument, **XInstallColormap** adds the colormap to the top of the list. If necessary, it truncates a colormap at the bottom of the list so that the maximum length of the list is not exceeded.
>
> If a colormap resource ID is passed to the *cmap* argument and this colormap is in the required list, **XUninstallColormap** removes the colormap from the required list.

A colormap is not implicitly added to the required list when it is installed by the server. Nor is a colormap in the required list implicitly uninstalled by the server.

Subtopics
1.7.6.1 Related Functions
1.7.6.2 Reading Entries in a Colormap
1.7.6.3 Related Functions

¦ Copyright IBM Corp. 1985, 1991

1.7.6.1 Related Functions

XInstallColorma

XUninstallColorma

XListInstalledColormap

¦ Copyright IBM Corp. 1985, 1991

1.7.6.2 Reading Entries in a Colormap

The **XQueryColor** and **XQueryColors** functions return the red, green, and blue color values stored in the specified colormap for the pixel value passed in the pixel member of the **XColor** data structures. The values returned for an unallocated entry are undefined. These functions also set the *flags* member in the **XColor** data structure to all three colors. See "Creating Colormaps" in topic 1.7.1.

¦ Copyright IBM Corp. 1985, 1991

1.7.6.3 Related Functions

XQueryColo

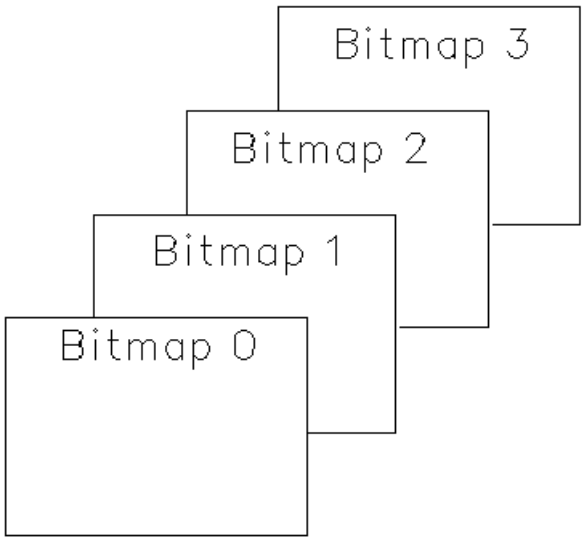
XQueryColor

¦ Copyright IBM Corp. 1985, 1991

1.7.7 Creating and Freeing Pixmaps

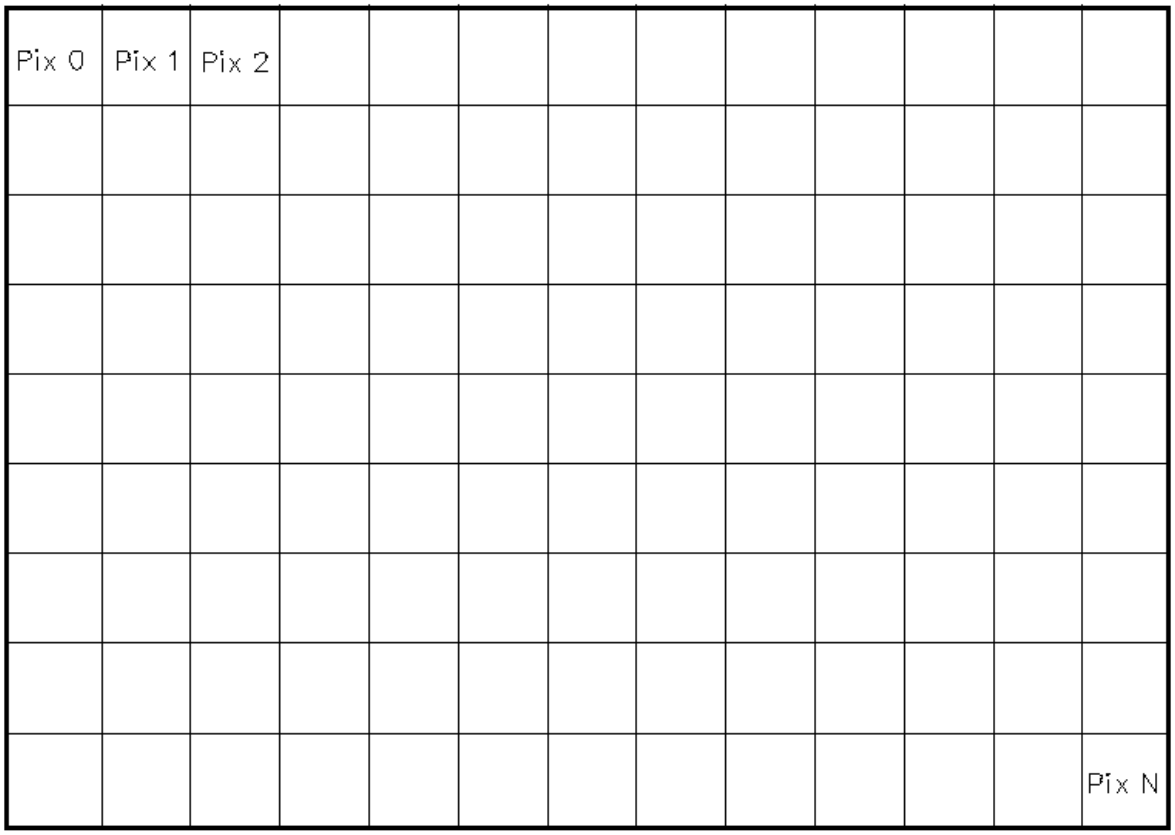
A ***pixmap*** is a rectangle of pixels that has the width and the height in pixels. A pixmap is as deep as the pixel has bits. Pixmaps are used only on the screen on which they are created. Pixmaps can be represented in storage in either ***XYformat*** or ***Zformat***.

In XYformat, each plane in the pixmap is represented as a bitmap; the bitmaps appear in storage from most significant to least significant in sequence. The following shows a pixmap in XY format:



In Zformat, the pixels are stored row by row, top to bottom, left to right within the row. The following shows a pixmap in Zformat:

¦ Copyright IBM Corp. 1985, 1991



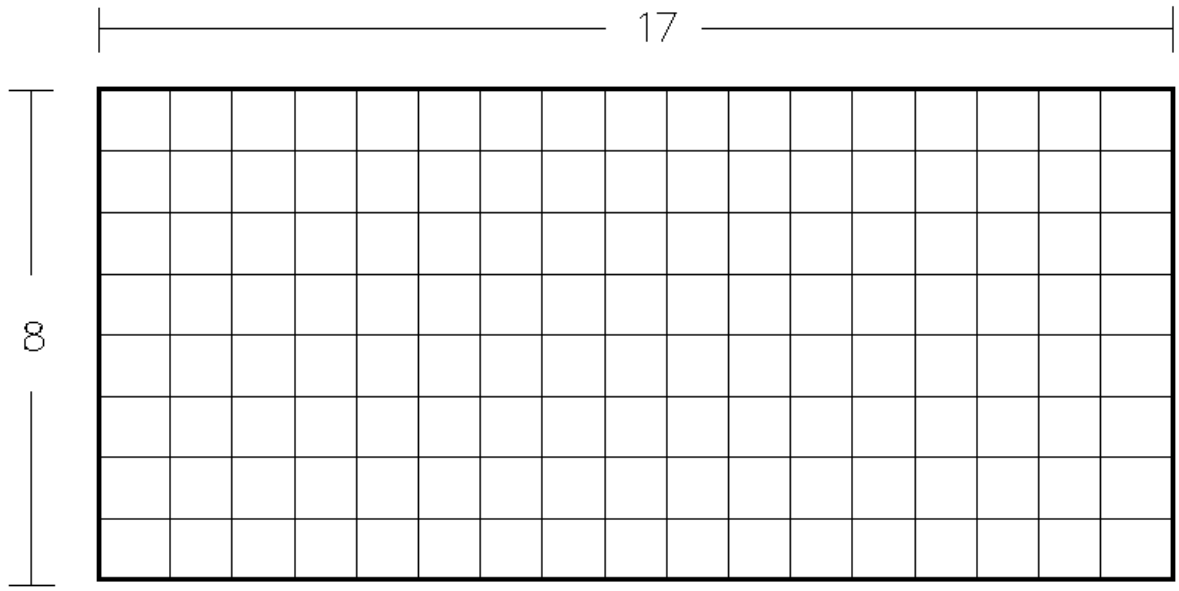
Xlib provides functions to create or free a pixmap. These operations include defining cursors as tiling patterns or as the source for certain raster operations. Most graphic requests can operate on a window or a pixmap. Some programs may want to manipulate pixels that are displayed on the screen later. Some functions move pixels from the program to the window system or from the window system to the program.

Subtopics
1.7.7.1 Defining Bitmaps
1.7.7.2 Related Functions

¦ Copyright IBM Corp. 1985, 1991

1.7.7.1 Defining Bitmaps

A ***bitmap*** is a rectangle of bits that has a width in pixels and a height in pixels. A bitmap is a pixmap of depth one. At depth one, Z format and XYformat are identical and are not specified in a bitmap. The following shows a bitmap 17 pixels wide by 8 pixels high.



¦ Copyright IBM Corp. 1985, 1991

1.7.7.2 Related Functions

XCreatePixma

XFreePixma

¦ Copyright IBM Corp. 1985, 1991

1.7.8 Manipulating Graphics Context or State

Most attributes of graphics operations are stored in graphics contexts (GCs). These attributes include line width, line style, plane mask, foreground, background, tile, stipple, clipping region, end style, join style, and others. Graphics operations (for example, drawing lines) use these values to determine the actual drawing operation. Extensions to X-Windows may add more components to GCs. **Xlib** provides calls for changing the state of GCs.

Xlib implements a write-back cache for all elements of a GC that are not resource IDs to allow it to implement the transparent coalescing changes to GCs. GCs are neither expected nor encouraged to be shared between client applications; therefore, this write-back caching should not present problems. Applications cannot share GCs without external synchronization; therefore, sharing GCs between applications is not encouraged.

The specified components of the new graphics context in *valuemask_create* are set to the values passed in the values argument. The other values default to the following values:

Component	**Value**	**Component**	**Value**
function	GXcopy	stipple	Pixmap of depth 1 of unspecified size filled with 1s
plane_mask	All 1s	ts_x_origin	0
foreground	0	ts_y_origin	0
background	1	font	Rom14.500 for large displays, Rom10.500 for small displays
line_width	0	subwindow_mode	ClipByChildren
line_style	LineSolid	graphics_exposures	True
cap_style	CapButt	clip_x_origin	0
join_style	JoinMiter	clip_y_origin	0
fill_style	FillSolid	clip_mask	None
fill_rule	EvenOddRule	dash_offset	0
arc_mode	ArcPieSlice	dashes	4 (the list [4,4])
tile	Pixmap of unspecified size filled with foreground pixel		

Use display functions to update a section of the destination screen with source bits from somewhere else. Many GC functions take one of these display functions as an argument. The function defines how the new destination bits are to be computed from the source bits and the old destination bits.

The **GXcopy** function is typically the most useful function because it works on a color display, but special applications can use other functions, particularly with certain planes of a color display. The functions, which are defined in **<X11/X.h>**, are:

Function Name	**Hex Code**	**Operation**
GXclear	0x0	0
GXand	0x1	source AND destination

¦ Copyright IBM Corp. 1985, 1991

GXandReverse	0x2	source AND NOT destination
GXcopy	0x3	source
GXandInverted	0x4	(NOT source) AND destination
GXnoop	0x5	destination
GXxor	0x6	source XOR destination
GXor	0x7	source OR destination
GXnor	0x8	(NOT source) AND (NOT destination)
GXequiv	0x9	(NOT source) XOR destination
GXinvert	0xa	NOT destination
GXorReverse	0xb	source OR (NOT destination)
GXcopyInverted	0xc	NOT source
GXorInverted	0xd	(NOT source) OR destination
GXnand	0xe	(NOT source) OR (NOT destination)
GXset	0xf	1

Note: Using display functions other than **GXcopy**, such as **GXxor** and **GXinvert**, on a color display can result in undefined pixel values.

For example, if an X Server on a four-plane display performs the **GXinvert** on pixel value **3**, the result is pixel value **12**. The color currently associated with pixel value **12** is displayed, whether or not it is a defined value.

Many of the color functions take either pixel values or planes as an argument. The planes argument specifies which planes of the display are to be modified (one bit per plane). A monochrome display has only one plane and this plane is the least-significant bit of the word. As planes are added to the display hardware, they occupy more significant bits in the plane mask. **AllPlanes**, a macro, can be used to refer to all planes of a display simultaneously.

Most operations use a GC. The contents of the GC are private to **Xlib**. Several procedures take structures of **XGCValues**. The following is a list of defined masks used to update a GC.

```
#define   GCFunction              (1L<<0)
#define   GCPlaneMask             (1L<<1)
#define   GCForeground            (1L<<2)
#define   GCBackground            (1L<<3)
#define   GCLineWidth             (1L<<4)
#define   GCLineStyle             (1L<<5)
#define   GCCapStyle              (1L<<6)
#define   GCJoinStyle             (1L<<7)
#define   GCFillStyle             (1L<<8)
#define   GCFillRule              (1L<<9)
#define   GCTile                  (1L<<10)
#define   GCStipple               (1L<<11)
#define   GCTileStipXOrigin       (1L<<12)
#define   GCTileStipYOrigin       (1L<<13)
#define   GCFont                  (1L<<14)
#define   GCSubwindowMode         (1L<<15)
#define   GCGraphicsExposures     (1L<<16)
#define   GCClipXOrigin           (1L<<17)
#define   GCClipYOrigin           (1L<<18)
#define   GCClipMask              (1L<<19)
#define   GCDashOffset            (1L<<20)
#define   GCDashList              (1L<<21)
#define   GCArcMode               (1L<<22)
```

¦ Copyright IBM Corp. 1985, 1991

The **XGCValues** data structure is defined as follows:

```
typedef struct {

  int function;                 /* logical operation                */
  unsigned long plane_mask;     /* plane mask                       */
  unsigned long foreground;     /* foreground pixel                 */
  unsigned long background;     /* background pixel                 */
  int line_width;               /* line width (in pixels)           */
  int line_style;               /* LineSolid, LineOnOffDash,        */
                                LineDoubleDash
  int cap_style;                /* CapNotLast, CapButt, CapRound,   */
                                CapProjecting
  int join_style;               /* JoinMiter, JoinRound, JoinBevel  */
  int fill_style;               /* FillSolid, FillTiled,            */
                                FillStippled, FillOpaqueStippled
  int fill_rule;                /* EvenOddRule, WindingRule         */
  int arc_mode;                 /* ArcChord, ArcPieSlice            */
  Pixmap tile;                  /* tile pixmap for tiling           */
                                operations
  Pixmap stipple;               /* stipple 1 plane pixmap for       */
                                stippling
  int ts_x_origin;              /* x offset for tile or stipple     */
                                operations
  int ts_y_origin;              /* y offset for tile or stipple     */
                                operations
  Font font;                    /* default text font for text       */
                                operations
  int subwindow_mode;           /* ClipByChildren,                  */
                                IncludeInferiors
  Bool graphics_exposures;      /* Boolean, should exposures be     */
                                generated
  int clip_x_origin;            /* x origin for clipping            */
  int clip_y_origin;            /* y origin for clipping            */
  Pixmap clip_mask;             /* bitmap clipping; other calls     */
                                for rects
  int dash_offset;              /* patterned/dashed line            */
                                information
  char dashes;

} XGCValues;
```

In graphics operations, given a source and destination pixel, the result is computed bitwise on corresponding bits of the pixels. A Boolean operation is performed in each bit plane. The *plane_mask* restricts the operations to a subset of planes. The result is computed by the following:

((src FUNC dst) AND plane_mask) OR (dst AND (NOT plane_mask))

Range checking is not performed on the values for *foreground*, *background*, or *plane_mask*. These values are truncated to the appropriate number of bits.

The *line_width* is measured in pixels. It can be greater than or equal to 1 (wide line) or can be the special value 0 (thin line).

Wide lines are drawn centered on the path described by the graphic

¦ Copyright IBM Corp. 1985, 1991

request.
Thin lines are drawn using an unspecified, device-dependent algorithm

A wide line drawn from [x1,y1] to [x2,y2] always draws the same pixels as a wide line drawn from [x2,y2] to [x1,y1] (excluding cap and join styles.) A *line_width* of 0 may differ from a *line_width* of 1 in which pixels are drawn.

In general, drawing a thin line is faster than drawing a wide line of width 1. However, because of their different drawing algorithms, thin lines may not mix well, aesthetically speaking, with wide lines. If it is desirable to obtain precise and uniform results across all displays, a client should always use a *line_width* of 1, rather than a *line_width* of 0.

The *line_style* defines which sections of a line are drawn:

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 (see fill-style) with **CapButt** style used where even and odd dashes meet.

LineOnOffDash Only the even dashes are drawn, and cap-style applies to all internal ends of the individual dashes, except **CapNotLast** is treated as **CapButt**.

The *cap_style* defines how the endpoints of a path are drawn:

CapNotLast Equivalent to **CapButt**, except that for a *line_width* of 0 or 1, the final endpoint is not drawn.

CapButt Square at the endpoint, perpendicular to the slope of the line, with no projection beyond.

CapRound A circular arc with the diameter equal to the *line_width*, centered on the endpoint (equivalent to **CapButt** for *line_width* 0 or 1).

CapProjecting Square at the end, but the path continues beyond the endpoint for a distance equal to half the *line_width* (equivalent to **CapButt** for *line_width* 0 or 1).

The *join_style* defines how corners are drawn for wide lines:

JoinMiter The outer edges of two lines extend to meet at an angle.

JoinRound A circular arc with diameter equal to the *line_width*, centered on the *joinpoint*.

JoinBevel **CapButt** endpoint styles, and then the triangular notch filled.

For a line with coincident endpoints (x1=x2, y1=y2), when the *cap_style* is applied to both endpoints, the semantics depends on the *line_width* and the

¦ Copyright IBM Corp. 1985, 1991

cap_style:

CapNotLast	thin	Nothing is drawn.
CapButt	thin	A single pixel is drawn.
CapButt	wide	Nothing is drawn.
CapRound	wide	The closed path is a circle, centered at the endpoint, with diameter equal to the *line_width*.
CapRound	thin	A single pixel is drawn.
CapProjecting	wide	The closed path is a square 4, aligned with the coordinate axes, centered at the endpoint, with sides equal to the *line_width*.
CapProjecting	thin	A single pixel is drawn.

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 was 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 when the *cap_style* is applied at both endpoints.

The *tile* or *stipple* and clip origins are interpreted relative to the origin of whatever destination drawable is specified in a graphics request.

The tile pixmap must have the same root and depth as the graphic context.
The stipple pixmap must have depth one and must have the same root a the GC. For stipple operations where the *fill_style* is **FillStippled**, but not **FillOpaqueStippled**, the stipple pattern is tiled in a single plane and acts as an additional clip mask to be ANDed with the *clip_mask*. Any size pixmap can be used for tiling or stippling although some sizes may be faster to use than others.

The *fill_style* defines the contents of the source for line, text, and fill requests. For all text and fill requests, for line requests with *line_style* **LineSolid**, and for the even dashes for line requests with *line_style* **LineOnOffDash** or **LineDoubleDash** the following applies:

FillSolid	Foreground.
FillTiled	Tile.
FillOpaqueStippled	A tile with the same width and height as stipple, but with stipple background has a 0 and with stipple foreground has a 1.
FillStippled	Foreground masked by stipple.

When drawing lines with *line_style* **LineDoubleDash**, the odd dashes are controlled by the *fill_style* in the following manner:

FillSolid	Background.
FillTiled	Same as even dashes.
FillOpaqueStippled	Same as even dashes.

¦ Copyright IBM Corp. 1985, 1991

FillStippled Background masked by stipple.

Storing a pixmap in a GC may result in a copy being made. If the pixmap is later used as the destination for a graphics request, the change may be reflected in the graphics context. If the pixmap is used simultaneously in a graphics request both as a destination and as a tile or stipple, the results are not defined.

The *dash_list* value allowed is a simplified form of the more general patterns that can be set with **XSetDashes**. Specifying a value of *N* is equivalent to specifying the two-element list *[N, N]* in **XSetDashes**. *N* specifies the length of the *dash_list*. This value must be nonzero. The default *dash_list* in a newly created GC is equivalent to [4,4].

The *dash_offset* variable specifies the phase of the pattern for the dashed line style to be set for the graphics context specifying how many elements into the *dash_list* the pattern should actually begin in any single graphics request.
The *dash_list* variable specifies the dash list for the dashed line style to be set for the specified GC.

Dashing is continuous through path elements combined with *join_style*, but is reset to the *dash_offset* each time a *cap_style* is applied at a line endpoint.

The unit of measure for dashes is the same as in the ordinary coordinate system. Ideally, a dash length is measured along the slope of the line, but implementations are required to match only for horizontal and vertical lines. It is suggested that you measure the length along the major axis of the line. The major axis is defined as the *x* axis for lines drawn at an angle of between -45 and +45 degrees or between 315 and 225 degrees from the *x* axis. For all other lines, the major axis is the *y* axis. The default *dash_list* in a newly created GC is equivalent to [4, 4].

The *clip_mask* restricts writes to the destination drawable.

If a pixmap is specified as the *clip_mask*, it must have a depth of one and have the same root as the GC.

If *clip_mask* is **None**, the pixels are always drawn, regardless of the clip origin.

The *clip_mask* can also be set with the **XSetClipRectangles** request. Only pixels where the *clip-mask* has a 1-bit are drawn. Pixels are not drawn outside the area covered by the *clip_mask* or where the *clip_mask* has a 0 bit. It affects all graphics requests.

The *clip_mask* does not clip sources.

The *clip_mask* origin is interpreted relative to the origin of the specified destination drawable in the graphics request.

For **ClipByChildren**, both source and destination windows are clipped additionally by all viewable **InputOutput** children.

For **IncludeInferiors**, neither the source window nor the destination window is clipped by inferiors. This results in drawing through the boundaries of subwindows. Using **IncludeInferiors** on a window with the depth of one with mapped inferiors of differing depth is allowed, but the semantics are

¦ Copyright IBM Corp. 1985, 1991

undefined by the core protocol.

The *fill_rule* defines which pixels are drawn inside for paths given in **XFillPolygon** requests. This variable can be set to the following:

EvenOddRule, which specifies a point is inside if an infinite ray with the point as origin crosses the path an odd number of times.
WindingRule, which specifies a point is inside if an infinite ray with the point as origin crosses an unequal number of clockwise and counterclockwise directed path segments.

- A clockwise directed path segment is one that crosses the ray from left to right as observed from the point.
- A counterclockwise segment is one that crosses the ray from right to left as observed from the point.

For **EvenOddRule** and **WindingRule**, a point is infinitely small, and the path is an infinitely thin line.

The *arc_mode* variable controls filling in the **XFillArcs** function. It can be one of the following:

ArcPieSlice, which specifies that arcs are pie-sliced filled.
ArcChord, which specifies that arcs are chord-filled.

The *graphics_exposures* variable controls the **GraphicsExpose** event generation for **XCopyArea** and **XCopyPlane** requests and any similar requests defined by extensions. The *graphics_exposures* events are sent even when they are not explicitly requested. To suppress them, set *graphics_exposures* to **False**.

Subtopics
1.7.8.1 Related Functions

¦ Copyright IBM Corp. 1985, 1991

1.7.8.1 Related Functions

XCreateGC
XFreeGC
XSetPlaneMask
XSetLineAttributes
XSetFillRule
XQueryBestStipple
XSetTSOrigin
XSetClipMask
XSetSubwindowMode

XCopyGC
XSetState
XSetForeground
XSetDashes
XQueryBestSize
XSetTile
XSetFont
XSetClipRectangles
XSetGraphicsExposures

XChangeGC
XSetFunction
XSetBackground
XSetFillStyle
XQueryBestTile
XSetStipple
XSetClipOrigin
XSetArcMode

¦ Copyright IBM Corp. 1985, 1991

1.8 Using Graphics Functions

Xlib graphics functions allow you to:

- Clear and copy area
- Draw points, lines, rectangles, and arc
- Fill area
- Manipulate font
- Draw text character
- Transfer images between clients and serve
- Manipulate cursors

Subtopics

¦ Copyright IBM Corp. 1985, 1991

1.8.1 Clearing Areas

Some **Xlib** functions allow you to clear an area or an entire drawable. **XClearArea** clears a specified rectangular area in a window, while **XClearWindow** clears the entire area.

Pixmaps do not have defined backgrounds; therefore, pixmaps cannot be filled by **XClearArea** or **XClearWindow**. Instead, use **XFillRectangle**, which sets the pixmap to a known value.

Subtopics
1.8.1.1 Related Functions

¦ Copyright IBM Corp. 1985, 1991

1.8.1.1 Related Functions

XClearAre

XClearWindo

¦ Copyright IBM Corp. 1985, 1991

1.8.2 Copying Areas

XLib functions copy an area or a bit-plane. When using **XCopyArea** to copy a specified area, the source drawable and the destination drawable must have the same root and the same depth. When using **XCopyPlane** to copy a single bit-plane to a drawable with depth >1, the bit-plane is color-expanded to the depth of the drawable.

In the source drawable, the one (1) bits cause the corresponding pixel in the destination drawable to be set to the foreground specified in the graphics context to alu the function and the planemask. In the same instance, the zero (0) bits in the source drawable cause the corresponding destination pixels to be set to the background color.

Subtopics
1.8.2.1 Related Functions

¦ Copyright IBM Corp. 1985, 1991

1.8.2.1 Related Functions

XCopyAre

XCopyPlan

¦ Copyright IBM Corp. 1985, 1991

1.8.3 Drawing Points, Lines, Rectangles, and Arcs

Xlib functions allow you to draw a single point or multiple points, a single line or multiple lines, a single rectangle or multiple rectangles, and a single arc or multiple arcs.

If the same drawable and GC is used for each call, **Xlib** batches back-to-back calls to **XDrawPoint**, **XDrawLine**, **XDrawRectangle**, **XFillArc**, and **XFillRectangle**.

The following **XSegment** data structure is used in the **XDrawSegments** function. See **XDrawSegments** in Chapter 2, "X-Windows Xlib Functions."

```
typedef struct {

    short x1, y1, x2, y2;

    } XSegment;
```

The following **XRectangle** data structure is used in the **XDrawRectangles** function. See **XDrawRectangles** in Chapter 2, "X-Windows Xlib Functions."

```
typedef struct {

    short x, y;
    unsigned short width, height;

    } XRectangle;
```

All *x* and *y* members are 16-bit signed integers. The *width* and *height* members are 16-bit unsigned integers. Do not generate coordinates and sizes out of the 16-bit ranges because the protocol has only 16-bit fields for these values. For example, the rectangle **{0,0,50000,1}** references the coordinates **x > = 49,999**, and **y = 0**. This cannot be represented in 16 bits and the results are not defined.

Subtopics
1.8.3.1 Related Functions
1.8.3.2 Drawing Single and Multiple Points
1.8.3.3 Related Functions
1.8.3.4 Drawing Single and Multiple Arcs
1.8.3.5 Related Functions

¦ Copyright IBM Corp. 1985, 1991

1.8.3.1 Related Functions

XDrawLine

XDrawLines

XDrawSegments

XDrawRectangle

XDrawRectangles

¦ Copyright IBM Corp. 1985, 1991

1.8.3.2 Drawing Single and Multiple Points

To draw single points, use **XDrawPoint**. This function uses the foreground pixel and the GC components to draw a single point into a drawable. This function is not affected by the tile or stipple in the GC. For more information, see **XDrawPoint** in Chapter 2, "X-Windows Xlib Functions."

To draw multiple points, use **XDrawPoints**. With this function, specify an array of **XPoint** data structures. **XDrawPoints** uses the foreground pixel and the GC components to draw multiple points into a drawable. This function is not affected by the tile or stipple in the GC. For more information, see **XDrawPoints** in Chapter 2, "X-Windows Xlib Functions."

The following **XPoint** data structure is used in the **XDrawPoints** function:

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

All *x* and *y* members are 16-bit signed integers. Do not generate coordinates and sizes out of the 16-bit range because the protocol has only 16-bit fields for these values.

¦ Copyright IBM Corp. 1985, 1991

1.8.3.3 Related Functions

XDrawPoin

XDrawPoint

¦ Copyright IBM Corp. 1985, 1991

1.8.3.4 Drawing Single and Multiple Arcs

To draw single and multiple arcs, use **XDrawArc** and **XDrawArcs**. **XDrawArc** draws a single circular or elliptical arc, while **XDrawArcs** draws multiple circular or elliptical arcs. Each arc drawn is specified by a rectangle and two angles.

The *x* and *y* coordinates of the rectangle are relative to the origin of the drawable. For an arc specified as **[x,y,w,h,a1,a2]**, the origin of the major and minor axes is at **[x+(w/2),y+(h/2)]**.

The infinitely thin path describing the entire circle or ellipse intersects the horizontal axis at **[x,y+(h/2)]** and **[x+w,y+(h/2)]** and the vertical axis at **[x+(w/2),y]** and **[x+(w/2),y+h]**. These coordinates can be fractional. The paths should be defined by the ideal mathematical path.

For a wide line with line-width *lw*, the bounding outlines for filling are given by the infinitely thin paths describing the arcs:

```
[x+dx/2, y+dy/2, w-dx, h-dy, a1, a2]

          and

[x-lw/2, y-lw/2, w+lw, h+lw, a1, a2]

          where

   dx=min(lw,w)
   dy=min(lw,h)
```

For an arc specified as **[x,y,w,h,a1,a2]**, the angles must be specified in the effectively skewed coordinate system of the ellipse; for a circle, the angles and coordinate systems are identical. The relationship between these angles and angles expressed in the normal coordinate system of the screen (as measured with a protractor) is as follows:

```
skewed-angle = atan(tan(normal-angle) * w/h) + adjust
```

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

```
0       for normal-angle in the range [0,PI/2)
PI      for normal-angle in the range [PI/2,(3*PI)/2)
2*PI    for normal-angle in the range [(3*PI)/2,2*PI)
```

The following **XArc** data structure is used in the **XDrawArcs** function.

```
typedef struct {
    short x, y;
    unsigned short width, height;
    short angle1, angle2;          /* Degrees multiplied by 64 */
    } XArc;
```

¦ Copyright IBM Corp. 1985, 1991

All *x* and *y* members are 16-bit signed integers. The *width* and *height* are 16-bit unsigned integers. Your application should not generate coordinates and sizes out of the 16-bit ranges because the protocol has only 16-bit fields for these values.

¦ Copyright IBM Corp. 1985, 1991

1.8.3.5 Related Functions

XDrawAr

XDrawArc

¦ Copyright IBM Corp. 1985, 1991

1.8.4 Filling Areas

To fill single rectanglar areas in a specified drawable, use **XFillRectangle**. To fill multiple rectangular areas in the specified drawable, use **XFillRectangles**. Both functions fill the specified rectangle or rectangles as if a four-point **XFillPolygon** were specified for each rectangle. The following is an example of a four-point **XFillPolygon**:

```
  [x,y] [x+width, y] [x+width, y+height] [x,y+height]
```

Both functions use *x* and *y* coordinates, *width* and *height* dimensions, and the GC specified.

Subtopics
1.8.4.1 Related Functions

¦ Copyright IBM Corp. 1985, 1991

1.8.4.1 Related Functions

XFillRectangl

XFillRectangle

XFillPolygo

XFillAr

XFillArc

¦ Copyright IBM Corp. 1985, 1991

1.9 Manipulating Fonts

Xlib provides functions that enable you to use and manipulate fonts. A ***font*** is a graphical family or assortment of characters of a given size or style.

The X Server loads fonts as they are requested by a program. The server can cache fonts for quick lookup. Fonts are global across all screens in a server. Fonts can be dealt with at several different levels. However, most applications simply use **XLoadQueryFont** to load a font and query the font metrics. See **XLoadQueryFont** in Chapter 2, "X-Windows Xlib Functions."

Characters in fonts are regarded as masks. Except for image text requests, the only pixels modified are pixels in which bits are on in the character. This means that it makes sense to draw text using stipples or tiles. For example, many menus gray-out unusable entries.

Subtopics

¦ Copyright IBM Corp. 1985, 1991

1.9.1 Defining Font Structures

X-Windows supports both single byte per character and two bytes per character text operations. Either form can be used with a font, but a single byte per character text request can specify a single byte only, that is, the first row of a two-byte font. A two-byte font is similar in concept to a two-dimensional matrix of defined characters.

byte1 Specifies the range of defined rows.

byte2 Defines the range of defined columns of the font.

Single byte per character fonts have no rows defined. The *byte2* range specified in the structure defines a range of characters.

The **XCharStruct** data structure is defined as:

```
typedef struct {
        short lbearing;                 /* origin to left edge of      */
                                        raster
        short rbearing;                 /* origin to right edge of     */
                                        raster
        short width;                    /* advance to next             */
                                        character's origin
        short ascent;                   /* baseline to top edge of     */
                                        raster
        short descent;                  /* baseline to bottom edge     */
                                        of raster
        unsigned short attributes;      /* per character flags (not    */
                                        predefined)

} XCharStruct;
```

XCharStruct defines the ***bounding box*** of a character. (The bounding box of a character is the smallest rectangle that encloses the shape of the character at the same *x, y* origin.)

The **XFontProp** data structure is defined as:

```
typedef struct {

    Atom  name;
    unsigned long CARD32;

} XFontProp;
```

The **XFontProp** data structure is a pointer to an array of additional properties for that font.

The **XChar2b** data structure is defined as:

```
        typedef struct {                /* normal 16-bit characters    */
                                        are 2 bytes
           unsigned char byte1;
           unsigned char byte2;

} XChar2b;
```

¦ Copyright IBM Corp. 1985, 1991

The **XChar2b** data structure is used in **Xlib** functions that use 2-byte character strings.

The **XFontStruct** structure contains all the information for the font, including font specific information and a pointer to an array of **XCharStruct** structures for the characters contained in the font. If characters are undefined or nonexistent, the *default_char* is used. If the font has characters all the same size, only the information in the **XFontStruct** characters is used. The **XFontStruct** data structure is defined as:

```
typedef struct {
        XExtData *ext_data;             /* hook for extension to hang   */
                                        data
        Font fid;                       /* Font ID for this font        */
        unsigned direction;             /* hint about the direction     */
                                        font is painted
        unsigned min_char_or_byte2;     /* first character              */
        unsigned max_char_or_byte2;     /* last character               */
        unsigned min_byte1;             /* first row that exists        */
        unsigned max_byte1;             /* last row that exists         */
        Bool all_chars_exist;           /* flag if all characters;      */
                                        have non-zero size
        unsigned default_char;          /* char to print for            */
                                        undefined character
        int n_properties;               /* how many properties there    */
                                        are
        XFontProp *properties;          /* pointer to array of          */
                                        additional properties
        XCharStruct min_bounds;         /* minimum bounds over all      */
                                        existing char
        XCharStruct max_bounds          /* maximum bounds over all      */
                                        existing char
        XCharStruct *per_char;          /* first_char to last_char      */
                                        information
        int ascent;                     /* logical extent above         */
                                        baseline for spacing
        int descent;                    /* logical decent below         */
                                        baseline for spacing
} XFontStruct;
```

The members of the **XFontStruct** data structure include the following:

The *direction* member which provides a hint as to what most **XCharStruct** elements have in terms of character-width metric. The *direction* member can be set to one of the following:

- **FontLeftToRight**, which signifies a positive character-width metric
- **FontRightToLeft**, which signifies a negative character-width metric.

The Core protocol does not support vertical text.

The *min_byte1* which indicates the first row that exists.

The *max_byte1* which indicates the last row that exists.

- If the *min_byte1* and *max_byte1* members are both zero, then

¦ Copyright IBM Corp. 1985, 1991

-- the *min_char_or_byte2* specifies the linear character index corresponding to the first element of the *per_char* array,

-- the *max_char_or_byte2* specifies the linear character index of the last element.

- If either *min_byte1* or *max_byte1* is non-zero, then both *min_char_or_byte2* and *max_char_or_byte2* will be less than 256.

The two-byte character index values corresponding to the *per_char* array element *N* (counting from 0) are:

```
byte1 = N/D + min_byte1
byte2 = N\D + min_char_or_byte2
```

where:

```
D = max_char_or_byte2 - min_char_or_byte2 + 1
/ = integer division
\ = integer modulus
```

If the *per_char* pointer is NULL, all glyphs between the first and last character, indexes inclusive, have the same information as given by both *min_bounds* and *max_bounds*.

If *all_chars_exist* is **True**, all characters in the *per_char* array have non-zero bounding boxes.

The *default_char* member specifies the character to use when an undefined or nonexistent character is used. The *default_char* is a 16-bit character. For a font using 2-byte matrix format, the *default_char* has *byte1* in the most-significant byte and *byte2* in the least-significant byte.

If the *default_char* specifies an undefined or nonexistent character, no printing is performed.

The *min_bounds* and *max_bounds* members contain the most extreme values of each individual **XCharStruct** component over all elements of this array which ignores nonexistent characters. The bounding box of the font is defined as follows:

- The upper-left coordinate is:

  [x + min_bounds.lbearing, y - max_bounds.ascent]

  The *x* and *y* coordinates are the lower-left corner of the box, relative to the origin.

- The width is:

  max_bounds.rbearing - min_bounds.lbearing

- The height is:

  max_bounds.ascent + max_bounds.descent

¦ Copyright IBM Corp. 1985, 1991

- The *ascent* member is the logical extent of the font above the baseline that is used for determining line spacing. Specific characters can extend beyond this.
- The *descent* member is the logical extent of the font at or below the baseline that is used for determining line spacing. Specific characters may extend beyond this.

  If the baseline is at Y-coordinate *y*, the logical extent of the font is inclusive between the Y-coordinate values **(y - font.ascent)** and **(y + font.descent - 1)**.

For a character origin at **[x, y]**, the bounding box of a character in terms of **XCharStruct** components, is a rectangle:

- The upper-left corner is:

  [x + lbearing, y - ascent]

- The width is:

  rbearing - lbearing

- The height is:

  ascent + descent

The origin for the next character is defined as:

[x + character-width, y]

The baseline is logically viewed as being immediately below non-descending characters. When *descent* is zero, only pixels with Y-coordinates less than *y* are drawn. The origin is logically viewed as being coincident with the left edge of a non-kerned character.

When *lbearing* is zero, no pixels with the X-coordinate less than *x* are drawn. Any of these values can be negative.

The interpretation of the attributes member in the **XCharStruct** structure is not defined by the core protocol. A nonexistent character is represented with members of the **XCharStruct** data structure set to zero.

A font is not guaranteed to have any properties. The interpretation of the property value, for example, **INT32**, **CARD32**, must be derived from a prior knowledge of the property. When possible, fonts should have the properties listed in the following table. (Atom names are case-sensitive.) The following built-in property atoms are in **<X11/Xatom.h>**.

Property Name	Type	Description
MIN_SPACE	unsigned	The minimum interword spacing (in pixels).
NORM_SPACE	unsigned	The normal interword spacing (in pixels).
MAX_SPACE	unsigned	The maximum interword spacing (in pixels).

¦ Copyright IBM Corp. 1985, 1991

END_SPACE	unsigned	The additional spacing (in pixels) at the end of sentences.
SUPERSCRIPT_X *SUPERSCRIPT_Y*	integer	Offset (in pixels) from character origin where superscripts should begin. If origin is at [*x,y*], then superscripts should begin at [*x* + SUPERSCRIPT_X, *y* - SUPERSCRIPT_Y].
SUBSCRIPT_X *SUBSCRIPT_Y*	integer	Offset (in pixels) from character origin where subscripts should begin. If origin is at [*x,y*], then subscripts should begin at [*x* + SUPERSCRIPT_X, *y* + SUPERSCRIPT_Y].
UNDERLINE_POSITION	integer	Y offset (in pixels) from the baseline to the top of an underline. If the baseline is Y-coordinate *y*, then the top of the underline is at (*y* + UNDERLINE_POSITION).
UNDERLINE_THICKNESS	unsigned	Thickness (in pixels) of the underline.
STRIKEOUT_ASCENT *STRIKEOUT_DESCENT*	integer	Vertical extents (in pixels) for boxing or voiding characters. If the baseline is at Y-coordinate *y*, then the top of the strikeout box is at **(***y* **- STRIKEOUT_ASCENT)**, and the height of the box is **(STRIKEOUT_ASCENT + STRIKEOUT_DESCENT)**.
ITALIC_ANGLE	integer	The angle of the dominant staffs of characters in the font, in degrees scaled by 64, relative to the three-o-clock position from the character origin, with positive indicating counterclockwise motion.
X_HEIGHT	integer	*1 ex* as in TeX, but expressed in pixels. Often the height of lowercase *x*.
QUAD_WIDTH	integer	*1 em* as in TeX, but expressed in pixels. Often the width of the digits 0-9.
CAP_HEIGHT	integer	Y offset (in pixels) from the baseline to the top of the capital letters, ignoring accents. If the baseline is at Y-coordinate *y*, then the top of the uppercase letters is at **(y - CAP_HEIGHT)**.
WEIGHT	unsigned	The weight or boldness of the

¦ Copyright IBM Corp. 1985, 1991

		font, expressed as a value between 0 and 1000.
POINT_SIZE	unsigned	The point size of the font at the ideal resolution, expressed in 1/10ths of points. There are 72.27 points to the inch.
RESOLUTION	unsigned	The number of pixels per point, expressed in 1/100ths, at which the font was created.

Subtopics
1.9.1.1 Related Functions

¦ Copyright IBM Corp. 1985, 1991

1.9.1.1 Related Functions

XLoadFont
XFreeFontInfo
XGetFontProperty
XFreeFontNames
XFreeFontPath
XTextExtents
XQueryTextExtents16

XGContextFromGC
XLoadQueryFont
XUnloadFont
XSetFontPath
XTextWidth
XTextExtents16

XListFontsWithInfo
XFreeFont
XListFonts
XGetFontPath
XTextWidth16
XQueryTextExtents

¦ Copyright IBM Corp. 1985, 1991

1.9.2 Drawing Text Characters

To draw 8-bit characters in the specified drawable, the text function **XDrawText** uses the **XTextItem** data structure:

```
typedef struct {
    char *chars;        /* pointer to string                          */
    int nchars;         /* number of characters                       */
    int delta;          /* delta between strings along the x axis     */
    Font font;          /* Font to print it in, None does not         */
                        change

} XTextItem;
```

To draw 2-byte characters in the specified drawable, the text function **XDrawText16** uses the **XTextItem16** data structure:

```
typedef struct {
    XChar2b *chars;     /*   pointer to two byte characters           */
    int nchars;         /*   number of characters                     */
    int delta;          /*   delta between strings along the x axis   */
    Font font;          /*   font to print it in, None does not       */
                            change

} XTextItem16;
```

If the *font* member is **None**, the font is changed before printing and is stored in the GC. If an error is generated during text drawing, the font in the GC is undefined.

The *chars* member of the **XTextItem16** data structure is of type **XChar2b**. The X Server interprets each member of the **XChar2b** structure as a 16-bit number that has been transmitted by most-significant byte first.

The *byte1* member of the **XChar2b** structure is taken as the most-significant byte.

¦ Copyright IBM Corp. 1985, 1991

1.9.3 Transferring Images Between Client and Server

Xlib functions transfer images between a client and the server. Because the server may require diverse data formats, X-Windows provides an image object that fully describes the data in memory and provides for basic operations on that data. The data should be referenced through the image object--not directly. However, some **Xlib** implementations can deal with frequently used data formats by replacing routines in the procedure vector with special case routines. These operations include destroying the image, getting a pixel, storing a pixel, extracting a subimage of an image, and adding a constant to an image.

Image manipulation functions use the following **XImage** data structure:

```
typedef struct _XImage {

   int width, height;          /* size of image                    */
   int xoffset;                /* number of pixels offset in X     */
                               direction
   int format;                 /* XYBitmap, XYPixmap, ZPixmap      */
   char *data;                 /* pointer to image data            */
   int byte_order;             /* data byte order, MSBFirst or     */
                               LSBFirst
   int bitmap_unit;            /* quantity of scanline             */
   int bitmap_bit_order;       /* MSBFirst or LSBFirst             */
   int bitmap_pad;             /* XYPixmap or ZPixmap              */
   int depth;                  /* depth of image                   */
   int bytes_per_line;         /* accelerator to next line         */
   int bits_per_pixel;         /* bits per pixel (ZPixmap)         */
   unsigned long red_mask;     /* bits in z arrangement            */
   unsigned long               /* bits in z arrangement            */
green_mask;
   unsigned long blue_mask;    /* bits in z arrangement            */
   char *obdata;               /* hook for the object routines to  */
                               hang on
   struct funcs {              /* image manipulation routines      */


          struct _XImage *(*create_image)();
          int (*destroy_image)();
          unsigned long (*get_pixel)();
          int (*put_pixel)();
          struct _XImage *(*sub_image)();
          int (*add_pixel)();
          } f;

} XImage;
```

The **XImage** data structure describes an image as it exists in client memory. You can request changes to some members in this data structure, for example, *height*, *width*, and *xoffset*. These changes create a subset of the image. Other members of this structure, for example, *byte_order* and *bitmap_unit*, are characteristic of both the image and the server. If these members differ between the image and the server, **XPutImage** makes the appropriate conversions.

If the image is formatted as an **XYPixmap**, the first byte of the first line of plane *n* must be located at the address of the client as follows:

(data + (n * height * bytes_per_line))

¦ Copyright IBM Corp. 1985, 1991

Subtopics
1.9.3.1 Related Functions

¦ Copyright IBM Corp. 1985, 1991

1.9.3.1 Related Functions

XPutImage

XGetImage

XGetSubImage

XCreateImage

XGetPixel

XPutPixel

XSubImage

XAddPixel

XDestroyImage

¦ Copyright IBM Corp. 1985, 1991

1.9.4 Manipulating Cursors

Some functions load and change cursors associated with windows. Each window can have a different cursor defined for it. Whenever the pointer is in a visible window, it will be set to the cursor defined for that window. If no cursor is defined for that window, the cursor is the cursor defined for the parent window.

For X-Windows, a ***cursor*** consists of a cursor shape, mask, colors for the shape and mask, and a hot spot. Client programs refer to cursors by using cursor resource IDs.

The *cursor* pixmap determines the shape of the cursor.

X-Windows provides a set of standard cursor shapes in a special font named *cursorfont*. Use this interface to customize your cursor for individual display. Use **XCreatePixmapCursor** to create a cursor from two bitmaps.

The *mask* pixmap determines the bits that are modified by the cursor.

The *colors* determines the colors of the shape and mask. The initial colors of a cursor are black foreground and white background. Use **XRecolorCursor** to change the color of the cursor.

The *hot spot* defines the point on the cursor that is reported when a pointer event occurs.

Hardware can impose limitations on cursor sizes and masks. (3) Use **XQueryBestCursor** to find out what size cursors are possible.

(3) AIX X-Windows supports cursor masks and cursor colors.

Subtopics
1.9.4.1 Related Functions

¦ Copyright IBM Corp. 1985, 1991

1.9.4.1 Related Functions

XCreateFontCursor

XCreateGlyphCursor

XRecolorCursor

XFreeCursor

XQueryBestCursor

XDefineCursor

XUndefineCursor

¦ Copyright IBM Corp. 1985, 1991

1.10 Using Window Manager Functions

Xlib window manager functions allow you to create a window manager application that can:

- Control the lifetime of a windo
- Manipulate the pointe
- Manipulate keyboard settings and the keyboard encodin
- Control host access

Subtopics

¦ Copyright IBM Corp. 1985, 1991

1.10.1 Controlling the Lifetime of a Window

The save-set of a window manager is a list of other client windows that should not be destroyed if the client windows are inferior windows of the window manager windows at connection close. To allow an application window to survive when a window manager fails, **Xlib** provides the save-set functions that change a window manager save-set, add a window to a window manager save-set, or remove a subwindow from a window manager save-set.

Some functions are used to control the longevity of subwindows that are normally destroyed when the parent is destroyed. For example, a window manager that wants to add decoration to a window by adding a frame might reparent an application window. When the frame is destroyed, the application window should not be destroyed but returned to its previous place in the window hierarchy.

The X Server automatically removes windows from the save-set when they are destroyed. If a save-set window is an inferior of a window manager window, the save-set window is reparented to the closest ancestor so that the save-set window is not an inferior window of a window created by the window manager. If the save-set window is unmapped, a **MapWindow** request is performed on it. After the save-set list is processed, all windows created by the window manager are destroyed. For each nonwindow resource created by the window manager, the appropriate **Free** request is performed. All colors and colormap entries allocated by the window manager are freed.

Subtopics
1.10.1.1 Related Functions

¦ Copyright IBM Corp. 1985, 1991

1.10.1.1 Related Functions

XChangeSaveSe

XAddToSaveSe

XRemoveFromSaveSe

¦ Copyright IBM Corp. 1985, 1991

1.10.2 Grabbing the Pointer

Xlib functions control input from the pointer, which is usually a mouse. The window manager uses these facilities to implement certain styles of user interface. Some applications may use these facilities for special purposes.

Usually, the X Server delivers keyboard and mouse events as soon as they occur to the appropriate client, depending upon the window and input focus. The X Server provides sufficient control over event delivery to allow window managers to support various styles of user interface. Many of the styles depend upon synchronous delivery of events. The delivery of pointer and keyboard events can be controlled independently.

When mouse buttons or keyboard keys are grabbed, events are sent to the grabbing client rather than the normal client. If the keyboard or pointer is in asynchronous mode, further mouse and keyboard events continue being processed. If the keyboard or pointer is in synchronous mode, no further events are processed until the grabbing client allows them to be processed. The keyboard or pointer is considered frozen during this interval. The triggering event can also be replayed.

There are two kinds of grabs: an active grab and a passive grab.

> An ***active grab*** occurs when a single client grabs the keyboard or pointer explicitly.
> A ***passive grab*** occurs when clients grab a particular keyboard key or pointer button in a window. Passive grabs are convenient for implementing reliable pop-up menus.

The grab activates when the key or button is actually pressed. For example, you can arrange that the pop-up is mapped before the *up* pointer button event occurs by grabbing a button requesting synchronous behavior. The *down* event triggers the grab and freezes further processing of pointer events until you have the chance to map the pop-up window. You can then allow further event processing. The *up* event is then correctly processed relative to the pop-up window.

For many operations, there are functions that take a *time* argument. The X Server includes a timestamp in various events. One special time called **CurrentTime** represents the current server time. The X Server maintains the time when the input focus was last changed and the time of the server when the client last performed an active grab, when the keyboard was last grabbed, or when a selection was last changed. This allows you to specify that your request not occur if some other application has in the meantime taken control of the keyboard, pointer, or selection.

Subtopics
1.10.2.1 Related Functions

¦ Copyright IBM Corp. 1985, 1991

1.10.2.1 Related Functions

XGrabPointer

XGrabButton

XUngrabButton

XUngrabPointer

XChangeActivePointerGrab

XGrabKeyboard

XUngrabKeyboard

XGrabKey

XUngrabkey

XAllowEvents

¦ Copyright IBM Corp. 1985, 1991

1.10.3 Grabbing the Server

Some **Xlib** functions grab and ungrab the server. These functions can be used to control processing of output on other connections by the window system server. No processing of requests or close downs on any other connection occurs while the server is grabbed. If a client closes its connection to the server, the client automatically ungrabs the server. Grabbing the server is highly discouraged unless it is absolutely necessary.

Subtopics
1.10.3.1 Related Functions

¦ Copyright IBM Corp. 1985, 1991

1.10.3.1 Related Functions

XGrabServe

XUngrabServe

¦ Copyright IBM Corp. 1985, 1991

1.10.4 Manipulating Keyboard Settings

With **Xlib** functions, you can change the keyboard control, obtain a list of the auto-repeat keys, turn keyboard auto-repeat on or off, ring the bell, set or obtain the pointer button or keyboard mapping, and obtain a bit vector for the keyboard.

You can set many of these options to your preference. The default values for many of these functions are determined by command line arguments to the X Server. Not all implementations can actually control all of these parameters.

ChangeKeyboardControl masks which are used in the **XKeyboardControl** data structure are defined as:

```
#define   KBKeyClickPercent       (1L<<0)
#define   KBBellPercent           (1L<<1)
#define   KBBellPitch             (1L<<2)
#define   KBBellDuration          (1L<<3)
#define   KBLed                   (1L<<4)
#define   KBLedMode               (1L<<5)
#define   KBKey                   (1L<<6)
#define   KBAutoRepeatMode        (1L<<7)
```

The **XKeyboardControl** data structure, which is used in functions that change control from a keyboard, is defined as:

```
typedef struct {

  int key_click_percent;
  int bell_percent;
  int bell_pitch;
  int bell_duration;
  int led;
  int led_mode;
  int key;
  int auto_repeat_mode;

} XKeyboardControl;
```

The members of the **XKeyboardControl** data structure include the following:

The *key_click_percent* member which sets the volume for key clicks between 0 (off) and 100 (loud) inclusive, if possible. A setting of -1 restores the default. Other negative values generate an error.

The *bell_percent* member which sets the base volume for the bell between 0 (off) and 100 (loud) inclusive, if possible. A setting of -1 restores the default. Other negative values generate an error.

The *bell_pitch* member which sets the pitch, specified in hertz (Hz) of the bell, if possible. A setting of -1 restores the default. Other negative values generate an error.

The *bell_duration* member which sets the duration of the bell (in milliseconds), if possible. A setting of -1 restores the default. Other negative values generate an error.

¦ Copyright IBM Corp. 1985, 1991

If the *led_mode* and the *led* members are specified, the state of those LEDs is changed, if possible. This occurs where LED is the ordinal number of the LED to be changed and not a mask.

If only *led_mode* is specified, the state of all LEDs is changed, if possible. At most 32 LEDs, numbered from 1, are supported. If an LED is specified without *led_mode*, an error is generated. No standard interpretation of LEDs is defined.

The *auto_repeat_mode* member which can be set to **AutoRepeatModeOff**, **AutoRepeatModeOn**, and **AutoRepeatModeDefault**.

- If the *auto_repeat_mode* and the *key* members are specified, the *auto_repeat_mode* of that key is changed, if possible.

- If only *auto_repeat_mode* is specified, the global *auto_repeat* mode for the entire keyboard is changed, if possible, and does not affect the *per_key* settings.

- If a key is specified without an *auto_repeat_mode*, an error is generated.

The order in which controls are verified and altered is server-dependent. If an error is generated, a subset of the controls may have been altered.

The following **XKeyboardState** data structure is used in functions that return the current control values for the keyboard:

```
typedef struct {

   int key_click_percent;
   int bell_percent;
   unsigned int bell_pitch, bell_duration;
   unsigned long led_mask;
   int global_auto_repeat;
   char auto_repeats[32];

   } XKeyboardState;
```

For the LEDs, the least-significant bit of *led_mask* corresponds to LED 1, and each bit in *led_mask* indicates an LED that is lit.

The *auto_repeat* member is a bit vector. Each bit indicates that auto-repeat is enabled for the corresponding key. The vector is represented as 32 bytes. Byte **N** (from 0) contains the bits for keys **8N to 8N+7**, with the least-significant bit in the byte representing key **8N**.

The *global_auto_repeat* member can be set to **AutoRepeatModeOn** or **AutoRepeatModeOff**.

Subtopics
1.10.4.1 Related Functions

¦ Copyright IBM Corp. 1985, 1991

1.10.4.1 Related Functions

XChangeKeyboardControl

XGetKeyboardControl

XAutoRepeatOn

XBell

XSetPointerMapping

XGetPointerMapping

XQueryKeymap

¦ Copyright IBM Corp. 1985, 1991

1.10.5 Manipulating Keyboard Encoding

A ***KeyCode*** represents a physical (or logical) key. Keycodes lie in the inclusive range [8,255]. A keycode value carries no intrinsic information. The mapping between keys and keycodes cannot be changed.

A ***KeySym*** is an encoding of a symbol on the cap of a key. The set of defined KeySyms includes the ISO Latin character sets (1-4), Katakana, Arabic, Cyrillic, Greek, Technical, Special, Publishing, APL, Hebrew, and a special miscellany of keys found on keyboards (RETURN, HELP, TAB, and so on). To the extent possible, these sets are derived from international standards. The list of defined symbols is in the **<X11/keysymdef.h>** header file. If you must use KeySyms not in the ISO Latin 1-4, Greek, and miscellany classes, you may have to define a symbol for those sets. Most applications usually include only **<X11/keysym.h>**, which defines symbols for ISO Latin 1-4, Greek, and miscellany.

A list of KeySyms is associated with each KeyCode. The length of the list can vary with each KeyCode. The list is intended to convey the set of symbols on the corresponding key. By convention, if the list contains a single KeySym and if that KeySym is alphabetic and case distinction is relevant for it, then it should be treated as equivalent to a two-element list of the lowercase and uppercase KeySyms. For example, if the list contains the single KeySym for uppercase *A*, the client should treat it as if it were a pair with lowercase *a* as the first KeySym and uppercase *A* as the second KeySym.

For any KeyCode, the first KeySym in the list should be chosen as the interpretation of a KeyPress when no modifier keys are down.

The second KeySym in the list normally should be chosen when the Shift modifier is on, or when the Lock modifier is on and Lock is interpreted as ShiftLock.

When the Lock modifier is on and is interpreted as CapsLock, it is suggested that the Shift modifier first be applied to choose a KeySym, but if that KeySym is lowercase alphabetic, the corresponding uppercase KeySym should be used instead.

Other interpretations of CapsLock are possible; for example, it may be viewed as equivalent to ShiftLock, but only applying when the first KeySym is lowercase alphabetic and the second KeySym is the corresponding uppercase alphabetic. No interpretation of KeySyms beyond the first two in a list is suggested here. No spatial geometry of the symbols on the key is defined by their order in the KeySym list although a geometry might be defined on a vendor-specific basis. The X Server does not use the mapping between KeyCodes and KeySyms. Rather, it stores the mapping for reading and writing by clients.

The interface, **XLookupString**, performs simple translation of a key event to an ASCII string.

The following **XModifierKeymap** data structure is used by functions that modify the keyboard mapping.

```
typedef struct {

   int max_keypermod;       /* Max number of keys per modifier of     */
                            this server
   KeyCode *modifiermap;    /* An 8 by max_keypermod array of the     */
```

¦ Copyright IBM Corp. 1985, 1991

} **XModifierKeymap;**

Subtopics
1.10.5.1 Related Functions

¦ Copyright IBM Corp. 1985, 1991

1.10.5.1 Related Functions

XGetKeyboardMapping

XChangeKeyboardMapping

XNewModifiermap

XInsertmodifiermapEntry

XDeleteModifiermapEntry

XFreeModifiermap

XSetModifierMapping

XGetModifierMapping

¦ Copyright IBM Corp. 1985, 1991

1.10.6 Controlling Host Access

X-Windows does not provide any protection on a per-window basis. If you find out the ID of a resource, you can manipulate it. To provide some minimal level of protection, however, connections are permitted only from systems you trust. This is adequate on single-user workstations, but breaks down on time-sharing machines.

The initial set of hosts allowed to open connections consists of:

The host the window system is running on

On AIX-based systems, the hosts listed in the **/etc/X?.hosts** file. The "?" indicates the number of the display. This file should consist of host names separated by newlines.

If a host is not in the access control list when the access control mechanism is enabled and if the host attempts to establish a connection, the server refuses the connection list or to change the access list. The client must reside on the same host as the server and/or must have been granted permission in the initial authorization at connection setup. The initial access control list can be specified by providing a file that the server can read at startup and reset time.

To add, get, or remove hosts, the host access control functions use the **XHostAddress** data structure. This data structure is defined as:

```
typedef struct {

   int family;             /* for example TCP/IP                    */
   int length;             /* length of address, in bytes           */
   char *address;          /* pointer to where to find the bytes    */

} XHostAddress;
```

The members of the **XHostAddress** data structure are:

family Specifies which protocol address family to use (for example, TCP/IP or UNIX-domain). The family symbols are defined in **<X11/X.h>**.

length Specifies the length of the address in bytes.

address Specifies a pointer to the address.

Subtopics
1.10.6.1 Related Functions

¦ Copyright IBM Corp. 1985, 1991

1.10.6.1 Related Functions

XAddHos

XAddHost

XListHost

XRemoveHos

XRemoveHost

¦ Copyright IBM Corp. 1985, 1991

1.11 Using Events and Event-Handling Functions

Most applications simply are event loops. That is, they wait for an event, decide what to do with it, execute some amount of code, which, in turn, results in changes to the display, and then wait for the next event. An ***event*** is data generated asynchronously by the X Server as a result of some device activity or as side effects of a request sent by an **Xlib** function.

A client application communicates with the X Server through the connection established with the **XOpenDisplay** function. A client application sends requests to the X Server through this connection. These requests are made by the **Xlib** functions that are called by the client applications. The X Server sends replies or events back to the client application. Most requests made by **Xlib** functions do not generate replies. Other requests generate multiple replies. Numerous **Xlib** functions cause the X Server to generate events. In addition, typing or moving the pointer can generate events asynchronously.

Subtopics

¦ Copyright IBM Corp. 1985, 1991

1.11.1 Defining Event Types

An event type describes a specific event generated by the X Server. For each event type, there is a corresponding constant name defined in **<X11/X.h>**. When referring to the event types, use the constant name defined in this file. Grouping one or more event types into logical categories is often useful. For example, exposure processing refers to the processing that occurs for the exposure events **Expose**, **GraphicsExpose**, and **NoExpose**.

Device-related events propagate from the source window to ancestor windows until some client application has selected that event type or until the event is explicitly discarded. The X Server sends an event to a client application only when the client application specifically asked to be informed of that event type, usually by calling the **XSelectInput** function. However, **KeymapNotify** events are always sent.

The following table lists the event categories and their associated event types.

Event Category	**Event Type**
Keyboard events	KeyPress KeyRelease
Pointer motion events	ButtonPress ButtonRelease MotionNotify
Window crossing events	EnterNotify LeaveNotify
Input focus events	FocusIn FocusOut
Key map state notification event	KeymapNotify
Exposure events	Expose GraphicsExpose NoExpose
Structure control events	CirculateRequest ConfigureRequest MapRequest ResizeRequest

¦ Copyright IBM Corp. 1985, 1991

Window state notification events	CirculateNotify ConfigureNotify CreateNotify DestroyNotify GravityNotify MapNotify MappingNotify ReparentNotify UnmapNotify VisibilityNotify
Colormap state notification event	ColormapNotify
Client communication events	ClientMessage PropertyNotify SelectionClear SelectionNotify SelectionRequest

¦ Copyright IBM Corp. 1985, 1991

1.11.2 Defining Event Structures

Each event type has a corresponding structure declared in the **<X11/Xlib.h>** file. All event structures have the following members:

type — Set to the event type constant name that uniquely identifies the event type. For example, when the X Server reports a **GraphicsExpose** event to a client application, the event sends an **XGraphicsExposeEvent** structure with the *type* member set to **GraphicsExpose**.

display — Set to a pointer to the display the event was read on.

send_event — Set to **True** if the event was generated by an **XSendEvent** request.

serial — Set to the serial number reported in the protocol but expanded from the 16-bit least-significant bits to a full 32-bit value.

The X Server may send events at any time in the input stream, even during the time the client application sends a request and receives a reply. **Xlib** stores, in an event queue for later use, any events received while waiting for a reply. It provides several functions that check events in the event queue.

In addition to the individual structures declared for each event type, there is also an **XEvent** structure. The **XEvent** structure is a union of the individual structures declared for each event type. After you determine the event type, use the structures declared in **<X11/Xlib.h>** when referring to it in a client application. Depending on the *type* member of the event, you should access elements of each event by using the **XEvent** structure.

¦ Copyright IBM Corp. 1985, 1991

1.11.3 Defining Event Masks

The event mask describes the event or events to be returned to the client application by the X Server. Client applications select event reporting of most events relative to a window. An event mask is passed in the *event_mask* argument to an **Xlib** event-handling function. The event masks are in the **<X11/X.h>** file.

Most events are not reported to clients when they are generated unless the client has specifically asked for them. However, **GraphicsExpose** and **NoExpose** events are reported by default as a result of **XCopyPlane** and **XCopyArea**, unless the client suppresses them by setting *graphics_expose* in the **GC** to **False**. **SelectionClear**, **SelectionRequest**, **SelectionNotify** or **ClientMessage** events cannot be masked, but generally are sent only to clients cooperating with selections. See "Using Window Selections" in topic 1.6.5. A **MappingNotify** event is always sent to clients when the keyboard mapping is changed.

The following table lists the event mask constants passed in the *event_mask* argument and the circumstances in which the event mask is used.

Event Mask	**Circumstances**
NoEventMask	No events wanted
KeyPressMask	Keyboard down events wanted
KeyReleaseMask	Keyboard up events wanted
ButtonPressMask	Pointer button down events wanted
ButtonReleaseMask	Pointer button up events wanted
EnterWindowMask	Pointer window entry events wanted
LeaveWindowMask	Pointer window leave events wanted
PointerMotionMask	Pointer motion events wanted
PointerMotionHintMask	Pointer motion hints wanted
Button1MotionMask	Pointer motion while button 1 down
Button2MotionMask	Pointer motion while button 2 down
Button3MotionMask	Pointer motion while button 3 down
Button4MotionMask	Pointer motion while button 4 down
Button5MotionMask	Pointer motion while button 5 down
ButtonMotionMask	Pointer motion while any button down
KeymapStateMask	Any keyboard state change wanted
ExposureMask	Any exposure wanted
VisibilityChangeMask	Any change in visibility wanted

Copyright IBM Corp. 1985, 1991

StructureNotifyMask	Any change in window structure wanted
ResizeRedirectMask	Redirect resize of this window
SubstructureNotifyMask	Substructure notification wanted
SubstructureRedirectMask	Redirect substructure of window
FocusChangeMask	Any change in input focus wanted
PropertyChangeMask	Any change in property wanted
ColormapChangeMask	Any change in Colormap wanted
OwnerGrabButtonMask	*owner_events* is True

¦ Copyright IBM Corp. 1985, 1991

1.11.4 Processing Events

The event types reported to a client application during event processing depend on which event masks are passed to the *event_mask* argument of the **XSelectInput** function. Some event masks have a one-to-one correspondence between the event mask constant and the event type constant. For example, if the event mask **ButtonPressMask** is passed, the X Server sends back only **ButtonPress** events. Most events contain a *time* member that indicates the time at which an event occurred.

In other cases, one event mask constant can map to several event type constants. For example, if the event mask **SubstructureNotifyMask** is passed, the X Server can send back **CirculateNotify**, **ConfigureNotify**, **CreateNotify**, **DestroyNotify**, **GravityNotify**, **MapNotify**, **ReparentNotify**, or **UnmapNotify** events.

In another case, two event mask constants map to one event type constant. For example, if the event mask **PointerMotionMask** or **PointerMotionHintMask** is passed, the X Server sends back a **MotionNotify** event.

The following table lists the event mask, the associated event type and the structure name associated with the event type. (NA appears in columns for which the information is not applicable.)

Event Mask	**Event Type**	**Structure**
ButtonMotionMask	MotionNotify	XPointerMovedEvent
Button1MotionMask	MotionNotify	XPointerMovedEvent
Button2MotionMask	MotionNotify	XPointerMovedEvent
Button3MotionMask	MotionNotify	XPointerMovedEvent
Button4MotionMask	MotionNotify	XPointerMovedEvent
Button5MotionMask	MotionNotify	XPointerMovedEvent
ButtonPressMask	ButtonPress	XButtonPressedEvent
ButtonReleaseMask	ButtonRelease	XButtonReleasedEvent
ColormapChangeMask	ColormapNotify	XColormapEvent
EnterWindowMask	EnterNotify	XEnterWindowEvent
ExposureMask	Expose	XExposeEvent
ExposureMask	GraphicsExpose	XGraphicsExposeEvent
ExposureMask	NoExpose	XNoExposeEvent
LeaveWindowMask	LeaveNotify	XLeaveWindowEvent
FocusChangeMask	FocusIn	XFocusInEvent
FocusChangeMask	FocusOut	XFocusOutEvent
KeymapStateMask	KeymapNotify	XKeymapEvent

Copyright IBM Corp. 1985, 1991

KeyPressMask	KeyPress	XKeyPressedEvent
KeyPressMask	KeyRelease	XKeyReleasedEvent
OwnerGrabButtonMask	NA	NA
PointerMotionMask	MotionNotify	XPointerMovedEvent
PointerMotionHintMask	MotionNotify	XPointerMovedEvent
PropertyChangeMask	PropertyNotify	XPropertyEvent
ResizeRedirectMask	ResizeRequest	XResizeRequestEvent
StructureNotifyMask	CirculateNotify	XCirculateEvent
StructureNotifyMask	ConfigureNotify	XConfigureEvent
StructureNotifyMask	DestroyNotify	XDestroyWindowEvent
StructureNotifyMask	GravityNotify	XGravityEvent
StructureNotifyMask	MapNotify	XMapEvent
StructureNotifyMask	ReparentNotify	XReparentEvent
StructureNotifyMask	UnmapNotify	XUnmapEvent
SubstructureNotifyMask	CirculateNotify	XCirculateEvent
SubstructureNotifyMask	ConfigureNotify	XConfigureEvent
SubstructureNotifyMask	CreateNotify	XCreateWindowEvent
SubstructureNotifyMask	DestroyNotify	XDestroyWindowEvent
SubstructureNotifyMask	GravityNotify	XGravityEvent
SubstructureNotifyMask	MapNotify	XMapEvent
SubstructureNotifyMask	ReparentNotify	XReparentEvent
SubstructureNotifyMask	UnmapNotify	XUnmapEvent
SubstructureRedirectMask	CirculateRequest	XCirculateRequestEvent
SubstructureRedirectMask	ConfigureRequest	XConfigureRequestEvent
SubstructureRedirectMask	MapRequest	XMapRequestEvent
NA	ClientMessage	XClientMessageEvent
NA	MappingNotify	XMappingEvent
NA	SelectionClear	XSelectionClearEvent
NA	SelectionNotify	XSelectionEvent
NA	SelectionRequest	XSelectionRequestEvent

Copyright IBM Corp. 1985, 1991

VisibilityChangeMask	VisibilityNotify	XVisibilityEvent

¦ Copyright IBM Corp. 1985, 1991

1.11.5 Processing Keyboard and Pointer Events

This section discusses the event processing that occurs when a pointer button is pressed and when the keyboard events **KeyPress** and **KeyRelease** and the pointer-motion events **ButtonPress**, **ButtonRelease**, and **MotionNotify** are generated.

Subtopics
1.11.5.1 Processing Specific Pointer Button Events
1.11.5.2 Processing Common Keyboard and Pointer Events

¦ Copyright IBM Corp. 1985, 1991

1.11.5.1 Processing Specific Pointer Button Events

When a pointer button is pressed with the pointer in a window and no active pointer grab is in progress, the X Server searches the ancestors of the specified window, from the root down, for a passive grab to activate. If there is no matching passive grab on the button, the X Server automatically starts an active grab for the client receiving the event and sets the last-pointer-grab time to the current server time. The effect is equivalent to an **XGrabButton** with the following client arguments:

window	The event window.
event_mask	The selected pointer-motion events of the client on the event window.
pointer_mode	**GrabModeAsync**.
keyboard_mode	**GrabModeAsync**.
owner_events	**True**, if the client selected **OwnerGrabButtonMask** on the event window. Otherwise, it is **False**.
confine_to	**None**.
cursor	**None**.

The grab is automatically terminated when all buttons are released. Clients can modify the active grab by calling **XUngrabPointer** and **XChangeActivePointerGrab**.

¦ Copyright IBM Corp. 1985, 1991

1.11.5.2 Processing Common Keyboard and Pointer Events

The X Server can report **KeyPress** and **KeyRelease** events to clients requesting information about when a key is pressed or released. The X Server generates these events whenever a key changes state, that is, whenever a key is pressed or released. These events are generated for all keys, even keys mapped to modifier bits.

The X Server reports **ButtonPress** and **ButtonRelease** events to clients requesting information about when a pointer button is pressed or released. The X Server generates these events whenever a pointer button changes state, that is, whenever a pointer button is pressed or released.

The X Server reports **MotionNotify** events to clients requesting information about when the pointer moves. The X Server generates this event whenever the pointer changes state, that is, whenever the pointer is moved and the pointer motion begins and ends in the window. The granularity of **MotionNotify** events is not guaranteed, but a client that selects this event type is guaranteed to receive at least one event when the pointer moves and then rests.

To receive **KeyPress**, **KeyRelease**, **ButtonPress**, and **ButtonRelease** events in a client application, you pass a window ID and **KeyPressMask**, **KeyReleaseMask**, **ButtonPressMask**, and **ButtonReleaseMask** as the *event_mask* arguments to **XSelectInput**.

To receive **MotionNotify** events in a client application, pass a window ID and one or more of the following event masks as the *event_mask* argument to **XSelectInput**:

Button1MotionMask-Button5MotionMas

> The client application receives **MotionNotify** events only when one or more of the specified buttons is pressed.

ButtonMotionMas

> The client application receives **MotionNotify** events only when at least one button is pressed.

PointerMotionMas

> The client application receives **MotionNotify** events independent of the state of the pointer buttons.

PointerMotionHintMas

> If **PointerMotionHintMask** is selected, the X Server is free to send only one **MotionNotify** event (with the *is_hint* member of the **XPointerMovedEvent** structure set to **NotifyHint**) to the client for the event window, until either the key or button state changes, or the pointer leaves the event window, or the client calls the **XQueryPointer** or **XGetMotionEvents** functions.

The source of the input event is the smallest window containing the pointer. The window used by the X Server to report these events depends on its position in the window hierarchy and whether any intervening window prohibits the generation of these events. The X Server searches up the window hierarchy, starting with the source window until it locates the first window specified by a client as having an interest in these events.

¦ Copyright IBM Corp. 1985, 1991

If one of the intervening windows has its *do_not_propagate_mask* set to prohibit generation of the event type, the event of those types will be suppressed. Clients can modify the actual window used for reporting by performing active grabs and, in the case of keyboard events, by using the focus window. The window in which the event is reported is the ***event window***.

The structures associated with these events are **XKeyPressedEvent**, **XKeyReleasedEvent**, **XButtonPressedEvent**, **XButtonReleasedEvent**, and **XPointerMovedEvent**. These structures have the following members:

The *window* member which is the window ID of the window on which the event was generated. This is the event window. The X Server uses this window to report the event.

The *root* member which is the window ID of the root window of the source.

The *x_root* and *y_root* members which are set to the pointer coordinates relative to the origin of the root window at the time of the event.

The *same_screen* member which indicates if the event window is on the same screen as the root window. This member can be **True** or **False**.

- If the event and root windows are on the same screen, *same_screen* is **True**.

- If the event and root windows are not on the same screen, *same_screen* is **False**.

The *subwindow* which can be one of the following:

- The child of the event window that is an ancestor of or is the source member, if the event window is on the same screen as the root window.

- Otherwise, the *subwindow* is **None**.

The *x* and *y* members which can be one of the following:

- The coordinates relative to the origin of the event window if the *source* window is an inferior of the event window.

- Otherwise, *x* and *y* are zero.

The *time* member which is the time that the event was generated. The time is expressed in milliseconds since the server reset.

The *state* member which indicates the state of the pointer buttons and modifier keys just prior to the event. The X Server can set this member to the bitwise inclusive-OR gate or operation of one or more of the following button or modifier key masks:

Button1Mask	**ShiftMask**	**Mod1Mask**
Button2Mask	**LockMask**	**Mod2Mask**
Button3Mask	**ControlMask**	**Mod3Mask**
Button4Mask		**Mod4Mask**
Button5Mask		**Mod5Mask**

¦ Copyright IBM Corp. 1985, 1991

The *detail* member which indicates one of the following:

- The *keycode* which is set to a number that represents a physical key on the keyboard for **XKeyPressedEvent** and **XKeyReleasedEvent**.

- The *button* which represents the pointer buttons that changed state for the **XButtonPressedEvent** and **XButtonReleasedEvent** structures. The *button* state can be set to the bitwise inclusive-OR gate of one or more of the following button names: **Button1**, **Button2**, **Button3**, **Button4**, **Button5**.

- The *is_hint* which can be set to **NotifyNormal** or **NotifyHint** for the **XPointerMovedEvent** structure.

¦ Copyright IBM Corp. 1985, 1991

1.11.6 Using Keyboard Utility Functions

Xlib provides functions to manipulate keyboard events or to determine information about a keysym. These functions include keyboard event functions.

Subtopics
1.11.6.1 Using Keyboard Event Functions
1.11.6.2 Related Functions

¦ Copyright IBM Corp. 1985, 1991

1.11.6.1 Using Keyboard Event Functions

The X Server does not predefine the keyboard to be ASCII characters. Knowing that the *a* key was just pressed or possibly that it was just released is often useful. When a key is pressed or released, the X Server sends keyboard events to client programs. The structures associated with keyboard events contain a *keycode* member that assigns a number to each physical key on the keyboard. See "Processing Keyboard and Pointer Events" in topic 1.11.5 for a discussion of keyboard event processing and "Manipulating Keyboard Encoding" in topic 1.10.5 for information on how to manipulate the keyboard encoding.

Because keycodes are arbitrary and may differ from server to server, client programs dealing with ASCII text, for example, must explicitly convert the keycode value into ASCII. The transformation of keycodes to ASCII or other character sets is arbitrary. Keyboards often differ and writing code that assumes the existence of a particular key on the main keyboard can create portability problems. It can also be difficult to receive **KeyRelease** events on certain X Server implementations because of hardware or software restrictions. Therefore, **Xlib** provides functions to customize the keyboard layout.

Keyboard events are usually sent to the smallest enclosing window that is interested in that type of event underneath the pointer position. It is also possible to assign the keyboard input focus to a specific window. When the input focus is attached to a window, keyboard events go to the client that selects input on that window rather than to the window under the pointer.

Some implementations cannot support **KeyRelease** events. While some applications exploit **KeyRelease** events to provide superior user interfaces, portability should be considered in designing software that uses **KeyRelease** events or changes key assignments. For example, it is possible to guarantee the existence of the a-z, spacebar, and carriage return keys, but not all keys on all keyboards.

¦ Copyright IBM Corp. 1985, 1991

1.11.6.2 Related Functions

XLookupKeysym

XRefreshKeyboardMapping

XLookupString

XRebindKeysym

XStringToKeysym

XKeysymToString

XKeycodeToKeysym

XKeysymToKeycode

IsKeypadKey macro

IsCursorKey macro

IsPFKey macro

IsFunctionKey macro

IsMiscFunctionKey macro

IsModifierKey macro

¦ Copyright IBM Corp. 1985, 1991

1.11.7 Processing Window Entry or Exit Events

This section describes the processing that occurs for the window crossing events. If a pointer motion or a window hierarchy change causes the pointer to be in a different window than before, the X Server reports **EnterNotify** or **LeaveNotify** events to clients that have selected these events.

EnterNotify and **LeaveNotify** events caused by a hierarchy change are generated after any hierarchy event, **UnmapNotify**, **MapNotify**, **ConfigureNotify**, **GravityNotify**, **CirculateNotify**, caused by that change; but the ordering of **EnterNotify** and **LeaveNotify** events with respect to **FocusOut**, **VisibilityNotify**, and **Expose** events is not constrained by the X protocol.

This type of processing contrasts with **MotionNotify** events, which are also generated when the pointer moves, but the pointer motion begins and ends in a single window. An **EnterNotify** or **LeaveNotify** event can also be generated when a client application calls any of the following structures: **XChangeActivePointerGrab**, **XGrabKeyboard**, **XGrabPointer**, and **XUngrabPointer**.

To receive **EnterNotify** events, pass a window ID and **EnterWindowMask** as the *event_mask* to **XSelectInput**.

To receive **LeaveNotify** events, pass the window ID and **LeaveWindowMask** as the *event_mask* to **XSelectInput**.

The members of the **XEnterWindowEvent** and **XLeaveWindowEvent** structures associated with these events are:

The *window* member which is set to the window ID of the window on which the **EnterNotify** or **LeaveNotify** event was generated. This window is the event window. The X Server uses this window to report the events.

The *root* member which is the ID of the root window on which the event occurred.

In a **LeaveNotify** event, if a child of the event window contains the initial position of the pointer, the *subwindow* is set to that child. Otherwise, the X Server sets the *subwindow* member to **None**.

In an **EnterNotify** event, if a child of the event window contains the final pointer position, the *subwindow* is set to that child. Otherwise, it is set to **None**.

The *time* member which is set to the time (in milliseconds) the event was generated.

The *x* and *y* members which are set to the pointer position in the event window. This position is always the final position of the pointer, not the initial position of the pointer.

If the event window is on the same screen as the root window, *x* and *y* are the pointer coordinates relative to the origin of the event window. Otherwise, *x* and *y* are set to zero.

The *x_root* and *y_root* members which are set to the pointer coordinates relative to the origin of the root window at the time of the event.

¦ Copyright IBM Corp. 1985, 1991

The *same_screen* member which indicates if the event window is on the same screen as the root window.

- If the event and root windows are on the same screen, *same_screen* is **True**.
- If the event and root windows are not on the same screen, *same_screen* is **False**.

The *focus* member which indicates whether the event window is the focus window or an inferior of the focus window.

- If the event window is the focus window or an inferior of the focus window, *focus* is **True**.

- If the event window is neither the focus window nor an inferior of the focus window, *focus* is **False**.

The *state* member which indicates the state of the pointer buttons and modifier keys immediately preceding the event. The X Server can set this member to the bitwise inclusive OR of one or more of the button or modifier key masks. See 1.11.5.2 for the values.

The *mode* member which indicates whether the events are normal events or pseudo-motion events when a grab activates, or when a grab deactivates. The X Server can set this member to **NotifyNormal**, **NotifyGrab**, or **NotifyUngrab**.

The *detail* member which indicates the notify detail can be set to one of the following:

NotifyAncestor **NotifyVirtual**

NotifyInferior **NotifyNonlinear**

NotifyNonlinearVirtual

¦ Copyright IBM Corp. 1985, 1991

1.11.8 Processing Normal Entry or Exit Events

EnterNotify and **LeaveNotify** events are generated when the pointer moves from one window to another window. Normal events are identified by **XEnterWindowEvent** or **XLeaveWindowEvent** structures with the *mode* member set to **NotifyNormal**.

When the pointer moves from window A to window B, and window A is an inferior of window B, the X Server generates:

A **LeaveNotify** event on window A that has the **XLeaveWindowEvent** structure with **NotifyAncestor** as the *detail* member.

A **LeaveNotify** event on each window between window A and window B exclusive, that has the **XLeaveWindowEvent** structure with **NotifyVirtual** as the *detail* member.

An **EnterNotify** event on window B that has the **XEnterWindowEvent** structure with **NotifyInferior** as the *detail* member.

When the pointer moves from window A to window B, and window B is an inferior of window A, the X Server generates:

A **LeaveNotify** event on window A that has the **XLeaveWindowEvent** structure with **NotifyInferior** as the *detail* member.

An **EnterNotify** event on each window between window A and window B exclusive, that has the **XEnterWindowEvent** structure with **NotifyVirtual** as the *detail* member.

An **EnterNotify** event on window B that has the **XEnterWindowEvent** structure with **NotifyAncestor** as the *detail* member.

When the pointer moves from window A to window B, and window C is their least common ancestor, the X Server generates:

A **LeaveNotify** event on window A that has the **XLeaveWindowEvent** structure with **NotifyNonlinear** as the *detail* member.

A **LeaveNotify** event on each window between window A and window C exclusive, that has the **XLeaveWindowEvent** structure with **NotifyNonlinearVirtual** as the *detail* member.

An **EnterNotify** event on each window between window C and window B exclusive, that has the **XEnterWindowEvent** structure with **NotifyNonlinearVirtual** as the *detail* member.

An **EnterNotify** event on window B that has the **XEnterWindowEvent** structure with **NotifyNonlinear** as the *detail* member.

When the pointer moves from window A to window B on different screens, the X Server generates:

A **LeaveNotify** event on window A that has the **XLeaveWindowEvent** structure with **NotifyNonlinear** as the *detail* member.

If window A is not a root window, the X Server generates a **LeaveNotify** event on each window above window A, up to and including its root, that has the **XLeaveWindowEvent** structure with **NotifyNonlinearVirtual** as the *detail* member.

¦ Copyright IBM Corp. 1985, 1991

If window B is not a root window, the X Server generates a **EnterNotify** event on each window from the root of window B, down to but not including window B, that has the **XEnterWindowEvent** structure with **NotifyNonlinearVirtual** as the *detail* member.

An **EnterNotify** event on window B that has the **XEnterWindowEvent** structure with the **NotifyNonlinear** as the *detail* member.

¦ Copyright IBM Corp. 1985, 1991

1.11.9 Processing Grab and Ungrab Entry or Exit Events

Pseudo-motion mode **EnterNotify** and **LeaveNotify** events are generated when a pointer grab activates or deactivates.

Events in which the pointer grab activates are identified b **XEnterWindowEvent** or **XLeaveWindowEvent** structures with the *mode* member set to **NotifyGrab**.

Events in which the pointer grab deactivates are identified b **XEnterWindowEvent** or **XLeaveWindowEvent** structures with the *mode* member set to **NotifyUngrab**.

The X Server generates **EnterNotify** and **LeaveNotify** events when a pointer grab activates after any initial warp into a *confine_to window,* before generating a **ButtonPress** event with G (*grab_window* for the grab) and P (the window the pointer is in), if **XEnterWindowEvent** and **XLeaveWindowEvent** structures have **NotifyGrab** as the *mode* member.

The **EnterNotify** and **LeaveNotify** events are generated as if the pointer warped suddenly from its current position in P to some position in G. However, the pointer does not warp. The X Server uses the pointer position as the initial and final positions for the events.

The X Server generates **EnterNotify** and **LeaveNotify** events when a pointer grab deactivates after generating a **ButtonRelease** event that deactivates the grab, with G (the *grab_window* for the grab) and P (the window the pointer is in) if the **XEnterWindowEvent** and **XLeaveWindowEvent** structures have **NotifyUngrab** as the *mode* member.

The **EnterNotify** and **LeaveNotify** events are generated as if the pointer warped suddenly from some position in G to its current position in P. However, the pointer does not warp. The X Server uses the current pointer position as the initial and final positions for the events.

¦ Copyright IBM Corp. 1985, 1991

1.11.10 Processing Input Focus Events

The X Server reports **FocusIn** or **FocusOut** events to client applications requesting information about changes to ***input focus*** or ***focus window***. The input focus or focus window is the point or window at which the keyboard is attached and input is received. The focus window can be the root window or a top-level window. The focus window and the position of the pointer determines which window receives keyboard input. Clients may need to know when the focus window changes. Input focus or the focus window changes when a client application uses **XGrabKeyboard** or **XUngrabKeyboard**.

To receive **FocusIn** and **FocusOut** events in a client application, pass a window ID and **FocusChangeMask** as the *event_mask* to **XSelectInput**.

The members of the **XFocusInEvent** and **XFocusOutEvent** structures include the following:

The *window* member which specifies the window ID of the window on which the **FocusIn** or **FocusOut** event was generated. The X Server uses this window to report the event.

The *mode* member which indicates whether the focus events are ***normal focus*** events or ***while grabbed focus*** events. Normal focus events are focus events generated when a grab activates. While grabbed focus events are focus events generated when a grab deactivates. The *mode* member can be set to one of the following:

NotifyNormal	**NotifyWhileGrabbed**
NotifyGrab	**NotifyUngrab**

The *detail* member which indicates the notify detail depending on the event mode. The *detail* member can be one of the following:

NotifyAncestor	**NotifyVirtual**
NotifyInferior	**NotifyNonlinear**
NotifyNonlinearVirtual	**NotifyPointer**
NotifyPointerRoot	**NotifyDetailNone**

All **FocusOut** events caused by a window unmap are generated after any **UnmapNotify** event, but the ordering of **FocusOut** events with respect to generated **EnterNotify**, **LeaveNotify**, **VisibilityNotify**, and **Expose** events is not constrained by the X protocol.

¦ Copyright IBM Corp. 1985, 1991

1.11.11 Processing Normal Focus and While Grabbed Focus Events

Normal focus events, which are events that occur when a grab activates, are identified by the **XFocusInEvent** or **XFocusOutEvent** structures with **NotifyNormal** as the *mode* member. While grabbed focus events, which are events that occur when a grab deactivates, are identified by **XFocusInEvent** or **XFocusOutEvent** structures with **NotifyWhileGrabbed** as the *mode* member. The X Server processes normal focus events and while grabbed focus events according to the following focus scenarios:

When the focus moves from window A to window B and window A is an inferior of window B with the pointer in window P, the X Server generates:

A **FocusOut** event on window A that has the **XFocusOutEvent** structure with **NotifyAncestor** as the *detail* member.

A **FocusOut** event on each window between window A and window B exclusive, that has the **XFocusOutEvent** structure with **NotifyVirtual** as the *detail* member.

A **FocusIn** event on window B that has the **XFocusOutEvent** structure with **NotifyInferior** as the *detail* member.

If window P is an inferior of window B, but window P is not window or an inferior of window A, the X Server generates a **FocusIn** event on each window below window B down to, and including, window P, that has a **XFocusInEvent** structure with **NotifyInferior** as the *detail* member.

When the focus moves from window A to window B, and window B is an inferior window of window A with the pointer in window P, the X Server generates:

If window P is an inferior of window A, but window P is not window or an inferior window of window B or an ancestor of window B, the X Server generates a **FocusOut** event on each window from window P up to, but not including, window A (in that order), that has the **XFocusOutEvent** structure with **NotifyPointer** as the *detail* member.

A **FocusOut** event on window A that has the **XFocusOutEvent** structure with **NotifyInferior** as the *detail* member.

A **FocusIn** event on each window between window A and window B exclusive, that has the **XFocusInEvent** structure with **NotifyVirtual** as the *detail* member.

A **FocusIn** event on window B that has the **XFocusInEvent** structure with **NotifyAncestor** as the *detail* member.

When the pointer moves from window A to window B, and window C is their least common ancestor, and the pointer is in window P, the X Server generates:

If window P is an inferior of window A, the X Server generates **FocusOut** event on each window from window P up to, but not including, window A, that has the **XFocusOutEvent** structure with **NotifyPointer** as the *detail* member.

A **FocusOut** event on window A that has the **XFocusOutEvent** structure with **NotifyNonlinear** as the *detail* member.

¦ Copyright IBM Corp. 1985, 1991

A **FocusOut** event on each window between window A and window C exclusive that has the **XFocusOutEvent** structure with **NotifyNonlinearVirtual** as the *detail* member.

A **FocusIn** event on each window between window C and window B exclusive that has the **XFocusInEvent** structure with **NotifyNonlinearVirtual** as the *detail* member.

A **FocusIn** event on window B that has the **XFocusInEvent** structure with **NotifyNonlinear** as the *detail* member.

If window P is an inferior of window B, the X Server generates **FocusIn** event on each window below window B down to, and including, window P, that has the **XFocusInEvent** structure with **NotifyPointer** as the *detail* member.

When the focus moves from window A to window B on different screens with the pointer in window P, the X Server generates:

If window P is an inferior of window A, the X Server generates **FocusOut** event on each window from window P up to, but not including, window A, that has the **XFocusOutEvent** structure with **NotifyPointer** as the *detail* member.

A **FocusOut** event on window A that has the **XFocusOutEvent** structure with **NotifyNonlinear** as the *detail* member.

If window A is not a root window, the X Server generates a **FocusOut** event on each window above window A, up to and including its root window, that has the **XFocusOutEvent** structure with **NotifyNonlinearVirtual** as the *detail* member.

If window B is not a root window, the X Server generates a **FocusIn** event on each window from the root of window B down to, but not including, window B, that has the **XFocusInEvent** structure with **NotifyNonlinearVirtual** as the *detail* member.

A **FocusIn** event on window B that has the **XFocusInEvent** structure with **NotifyNonlinear** as the *detail* member.

If window P is an inferior of window B, the X Server generates **FocusIn** event on each window below window B down to, and including, window P that has the **XFocusInEvent** structure with **NotifyPointer** as the *detail* member.

If the focus window was specified as **PointerRoot** (events sent to the window under the pointer) or **None** (discards the event) in **XSetInputFocus**, when the focus moves from window A to **PointerRoot** or **None** with the pointer in window P, the X Server:

If window P is an inferior of window A, the X Server generates **FocusOut** event on each window from window P up to, but not including, window A that has the **XFocusOutEvent** structure with **NotifyPointer** as the *detail* member.

A **FocusOut** event on window A that has the **XFocusOutEvent** structure with **NotifyNonlinear** as the *detail* member.

If window A is not a root window, the X Server generates a **FocusOut**

¦ Copyright IBM Corp. 1985, 1991

event on each window above window A up to, and including, its root, that has the **XFocusOutEvent** structure with **NotifyNonlinearVirtual** as the *detail* member.

A **FocusIn** event on the root window of all screens that has the **XFocusInEvent** structure with **NotifyPointerRoot** or **NotifyDetailNone** as the *detail* member.

If the new focus window is **PointerRoot**, the X Server generates a **FocusIn** event on each window from the root window of window P down to, and including window P that has the **XFocusInEvent** structure with **NotifyPointerRoot** as the *detail* member.

When the focus moves from **PointerRoot** or **None** to window A with the pointer in window P, the X Server generates:

If the old focus is **PointerRoot**, the X Server generates a **FocusOut** event on each window from window P up to, and including the root window of window P that has the **XFocusOutEvent** structure with **NotifyPointerRoot** as the *detail* member.

A **FocusOut** event on all root windows that have the **XFocusOutEvent** structure with **NotifyPointerRoot** or **NotifyDetailNone** as the *detail* member.

If window A is not a root window, the X Server generates a **FocusIn** event on each window from the root of window A down to, but not including, window A that has the **XFocusInEvent** structure with **NotifyNonlinearVirtual** as the *detail* member.

A **FocusIn** event on window A that has the **XFocusInEvent** structure with **NotifyNonlinear** as the *detail* member.

If window P is an inferior of window A, the X Server generates **FocusIn** event on each window below window A down to, and including, window P that has the **XFocusInEvent** structure with **NotifyPointer** as the *detail* member.

When the focus moves from **PointerRoot** to **None**, or vice versa, with the pointer in window P, the X Server generates:

If the old focus is **PointerRoot**, the X Server generates a **FocusOut** event on each window from window P up to, and including the root of window P that has the **XFocusOutEvent** structure with **NotifyPointerRoot** as the *detail* member.

Generates a **FocusOut** event on all root windows that have the **XFocusOutEvent** structure with **NotifyPointerRoot** or **NotifyDetailNone** as the *detail* member.

A **FocusIn** event on all root windows that have the **XFocusInEvent** structure with **NotifyDetailNone** or **NotifyPointerRoot** as the *detail* member.

If the new focus is **PointerRoot**, the X Server generates a **FocusIn** event on each window from the root of window P down to, and including, window P that has the **XFocusInEvent** structure with **NotifyPointerRoot** as the *detail* member.

¦ Copyright IBM Corp. 1985, 1991

1.11.12 Processing Focus Events Generated by Grabs

Focus events in which the keyboard grab activates are identified by **XFocusInEvent** or **XFocusOutEvent** structures whose *mode* member is **NotifyGrab**.

When a keyboard grab activates before generating an actual **KeyPress** event to the grab with window G (the *grab_window*) and to window F (the current focus window), the X Server generates **FocusIn** and **FocusOut** events if the **XFocusInEvent** and **XFocusOutEvent** structures have **NotifyGrab** as the *mode* members. These events are generated as if the focus had changed from window F to window G.

Focus events in which the keyboard grab deactivates are identified by **XFocusInEvent** or **XFocusOutEvent** structures whose *mode* member is **NotifyUngrab**.

When a keyboard grab deactivates after generating an actual **KeyRelease** event that deactivates the grab with window G (the *grab_window*) and window F (the current focus window), the X Server generates **FocusIn** and **FocusOut** events if **XFocusInEvent** and **XFocusOutEvent** structures have **NotifyUngrab** as the *mode* members. These events are generated as if the focus had changed from window G to window F.

¦ Copyright IBM Corp. 1985, 1991

1.11.13 Processing Keymap State Notification Events

The X Server reports **KeymapNotify** events to clients requesting information about changes in the keyboard state. To receive **KeymapNotify** events in a client application, pass a window ID and **KeymapStateMask** as the *event_mask* to **XSelectInput**. The X Server generates this event immediately after every **EnterNotify** and **FocusIn** event.

The members of the **XKeymapEvent** structure associated with this event include the following:

The *window* member which is not used, but is present for use with toolkit operations.

The *key_vector* member which specifies the bit vector of the keyboard. Each bit indicates that the corresponding key is currently pressed. The vector is represented as 32 bytes. **Byte N** (from 0) contains the bits for keys **8N to 8N+7** with the least-significant bit in the byte representing key **8N**.

¦ Copyright IBM Corp. 1985, 1991

1.11.14 Processing Exposure Events

The X protocol does not guarantee to preserve the contents of window regions when the windows are obscured or reconfigured. Some implementations can preserve the contents of windows, but many other implementations destroy the contents of windows when the windows are exposed. Client applications should be designed to restore the contents of an ***exposed window region***. (An exposed window region describes a formerly obscured window whose region or piece of a region becomes visible.) The X Server sends exposure events describing the window and the region of the window that has been exposed. Some client applications redraw the entire window, while other client applications redraw only the exposed region.

Subtopics
1.11.14.1 Processing Expose Events
1.11.14.2 Processing GraphicsExpose and NoExpose Events

¦ Copyright IBM Corp. 1985, 1991

1.11.14.1 Processing Expose Events

The X Server reports **Expose** events to clients requesting information about when the contents of the window regions have been lost. The X Server generates **Expose** events when a window region becomes viewable. It reports **Expose** events contiguously when all the regions are exposed by some action, such as raising a window. The X Server cannot generate **Expose** events to **InputOnly** windows.

To receive **Expose** events in a client application, pass a window ID and **ExposureMask** as the *event_mask* to **XSelectInput**.

The members of the **XExposeEvent** structure associated with this event include the following:

The *window* member which is the window ID of the exposed window.

The *x* and *y* members which are the coordinates that indicate the upper-left corner of the rectangle. These coordinates are set relative to the origin of the drawable.

The *width* and *height* members which specify the size (extent) of the rectangle.

The *count* member which specifies the number of **Expose** events that should follow.

- If *count* is zero, no **Expose** events will follow. (Applications not designed to optimize re-display by distinguishing between subareas of a window re-display entirely if *count* is zero.)

- If *count* is nonzero, at least that number, and possibly more, **Expose** events will follow. (Applications not designed to optimize re-display by distinguishing between subareas of a window, do not respond if *count* is nonzero.)

¦ Copyright IBM Corp. 1985, 1991

1.11.14.2 Processing GraphicsExpose and NoExpose Events

The X Server reports **GraphicsExpose** events to clients requesting information when a destination region could not be computed during a graphics request. Clients initiate graphics request with the **XCopyArea** or **XCopyPlane** functions.

The X Server generates a **GraphicsExpose** event whenever a destination region cannot be completed because of an obscured or out-of-bounds source region. The X Server reports contiguously all exposures of regions of a graphics request (for example, copying an area of a drawable to a destination drawable).

The X Server generates **NoExpose** events whenever a graphics request that might produce a **GraphicsExpose** event does not produce any graphics events. In other words, the client requests **GraphicsExpose** event, but receives a **NoExpose** event instead.

To receive **GraphicsExpose** or **NoExpose** events in a client application, set the *graphics_exposures* member of the **XGCValues** structure to **True**. (The *graphics_exposures* is set when using **XCreateGC** or **XSetGraphicsExposures**.)

The **XGraphicsExposeEvent** and **XNoExposeEvent** structures, which are associated with these event types, have the following members:

The *drawable* member which specifies the drawable ID of the destination region on which the copy request is to be performed.

The *major_code* member which specifies the graphics request initiated by the client can be **X_CopyArea** or **X_CopyPlane**. These constants are defined in **<X11/Xproto.h>**.

The *minor_code* member which specifies the graphics request initiated by the client is zero. Even though this member is not defined by the core X protocol, it can be used by an extension.

XGraphicsExposeEvent has the following additional members:

The *x* and *y* members which specify the upper-left corner of the rectangle. These coordinates are relative to the origin of the drawable.

The *width* and *height* members which specify the size (extent) of the rectangle.

The *count* member which specifies the number of **GraphicsExpose** events to follow.

- If *count* is zero, no **GraphicsExpose** events will follow.
- If *count* is nonzero, at least that number, and possibly more, **GraphicsExpose** events will follow.

¦ Copyright IBM Corp. 1985, 1991

1.11.15 Processing Window State Notification Events

Different processing occurs for window state notification events, such as **CirculateNotify**, **ConfigureNotify**, **CreateNotify**, **DestroyNotify**, **GravityNotify**, **MapNotify**, **MappingNotify**, **ReparentNotify**, **UnmapNotify**, and **VisibilityNotify**.

Subtopics

¦ Copyright IBM Corp. 1985, 1991

1.11.15.1 Processing CirculateNotify Events

The X Server reports **CirculateNotify** events to clients requesting information about when a window changes its position in the stack. The X Server generates this event type whenever a window is actually restacked as a result of a client application calling **XCirculateSubwindows**, **XCirculateSubwindowsUp**, or **XCirculateSubwindowsDown**.

To receive a **CirculateNotify** event in a client application, pass the window ID or the parent window ID and **StructureNotifyMask** as the *event_mask* to **XSelectInput**.

The members of the **XCirculateEvent** structure associated with this event include the following:

The *event* member which specifies the window ID of the window on which the **CirculateNotify** event was generated. The X Server uses this window to report the event.

The *window* member which specifies the window ID of the window that was restacked.

The *place* member which is the window position after the restack occurs.

- If *place* is **PlaceOnTop**, the window is now on top of all siblings.

- If *place* is **PlaceOnBottom**, the window is now below all siblings.

¦ Copyright IBM Corp. 1985, 1991

1.11.15.2 Processing ConfigureNotify Events

The X Server reports **ConfigureNotify** events to clients requesting information about actual changes to the state of a window, for example, size, position, border, and stacking order. It generates a **ConfigureNotify** event when a client appliation completes one of the following configure window requests:

XConfigureWindow which reconfigures window size, position, border, and stacking order.

XLowerWindow, **XRaiseWindow**, or **XRestackWindows** which change the window position in the stacking order.

XMoveWindow which moves the window.

XResizeWindow which changes the window size.

XMoveResizeWindow which changes the size and location of the window.

XMapRaised which changes the position of a mapped window in the stacking order.

XSetWindowBorderWidth which changes the border width of a window.

To receive a **ConfigureNotify** event, do one of the following:

Pass the window ID and **StructureNotifyMask** as the *event_mask* to **XSelectInput**.

Pass the parent window ID and **SubstructureNotifyMask** as the *event_mask* to **XSelectInput**.

The members of the **XConfigureEvent** structure associated with this event include the following:

The *event* member which specifies the window ID of the window on which the **ConfigureNotify** event was generated. The X Server uses this window to report the event.

The *window* member which specifies the window ID of the window whose size, position, border, or stacking order was changed.

The *x* and *y* members which specify the position of the upper-left outside corner of the window. These coordinates are relative to the origin of the new parent window.

The *width* and *height* members which specify the size of the window, excluding the border.

The *border_width* member which specifies the width (in pixels) of the window border.

The *above* member which specifies the window ID of the sibling window and is used for stacking operations.

- If *above* is **None**, the window whose state was changed is on the bottom of the stack with respect to sibling windows.

¦ Copyright IBM Corp. 1985, 1991

- If *above* is a sibling window, the window whose state was changed is on top of this sibling window.

The *override_redirect* member which specifies the *override_redirect* value of the **XSetWindowAttributes** structure when the window was created or the window attributes were changed. This member can be **True** or **False**. If *override_redirect* is **True**, client applications normally should ignore this event.

¦ Copyright IBM Corp. 1985, 1991

1.11.15.3 Processing CreateNotify Events

The X Server reports **CreateNotify** events to clients requesting information about creation of windows. The X Server generates this event when a client application creates a window with **XCreateWindow** or **XCreateSimpleWindow**.

To receive the **CreateNotify** event, pass the window ID of the parent window and **SubstructureNotifyMask** as the *event_mask* to **XSelectInput**.

The members of the **XCreateWindowEvent** structure associated with this event include the following:

The *parent* member which is the window ID of the parent of the created window.

The *window* member which specifies the window ID of the created window.

The *x* and *y* members which are the coordinates of the created window and which indicate the upper-left outside corner of the created window. These coordinates are relative to the inside of the borders of the parent window.

The *width* and *height* members which are set to the size of the created window, excluding the border. These dimensions are always a nonzero value.

The *border_width* member which specifies the width (in pixels) of the border of the created window.

The *override_redirect* member which specifies the *override_redirect* value of the **XSetWindowAttributes** structure when the window was created or the window attributes were changed. This member can be **True** or **False**. If *override_redirect* is **True**, client applications normally should ignore this event.

¦ Copyright IBM Corp. 1985, 1991

1.11.15.4 Processing DestroyNotify Events

The X Server reports **DestroyNotify** events to clients requesting information about windows that are destroyed. It generates this event whenever a client application destroys a window with **XDestroyWindow** or **XDestroySubwindows**.

A **DestroyNotify** event is generated on all inferiors of a window before the window is destroyed. The ordering among siblings and across subhierarchies is not constrained otherwise by the X protocol.

To receive a **DestroyNotify** event, pass the window ID of the parent window or the window ID and **SubstructureNotifyMask** as the *event_mask* to **XSelectInput**.

The following members of the **XDestroyWindowEvent** structure are associated with the **DestroyNotify** event:

The *event* member which specifies the window ID of the window on which the **DestroyNotify** event was generated. The X Server uses this window to report the event.

The *window* member which specifies the window ID of the window that is destroyed.

¦ Copyright IBM Corp. 1985, 1991

1.11.15.5 Processing GravityNotify Events

The X Server reports **GravityNotify** events to clients requesting information when a window is moved because of a change in the size of the parent window. The X Server generates this event when a client application moves a child window as a result of **XConfigureWindow**, **XMoveResizeWindow**, or **XResizeWindow**.

To receive **GravityNotify** events, do one of the following:

Pass the window ID of the window and **StructureNotifyMask** as the *event_mask* to **XSelectInput**.

Pass the window ID of the parent window and **SubstructureNotifyMask** as the *event_mask* to **XSelectInput**.

The following members of the **XGravityEvent** structure are associated with this event:

The *event* member which specifies the window ID of the window on which the **GravityNotify** event was generated. The X Server uses this window to report the event.

The *window* member which specifies the window ID of the child window that was moved.

The *x* and *y* members which indicate the position of the upper-left outside corner of the window. These coordinates are relative to the origin of the new parent window.

¦ Copyright IBM Corp. 1985, 1991

1.11.15.6 Processing MapNotify Events

The X Server reports **MapNotify** events to clients requesting information when windows are mapped. It generates this event type whenever a client application changes the state of a window from unmapped to mapped with **XMapWindow**, **XMapRaised**, or **XMapSubwindows**.

To receive **MapNotify** events, do one of the following:

Pass the window ID of the window and **StructureNotifyMask** as the *event_mask* argument to **XSelectInput**.

Pass the ID of the parent window and **SubstructureNotifyMask** as the *event_mask* argument to **XSelectInput**.

The following members of the **XMapEvent** structure are associated with this event:

The *event* member which specifies the window ID of the window on which the **MapNotify** event was generated. The X Server uses this window to report the event.

The *window* member which specifies the window ID of the window that was mapped.

The *override_redirect* member which specifies the *override_redirect* value of the **XSetWindowAttributes** structure when the window was created or when the window attributes were changed. This member can be **True** or **False**. If *override_redirect* is **True**, clients normally should ignore this event.

¦ Copyright IBM Corp. 1985, 1991

1.11.15.7 Processing MappingNotify Events

The X Server reports **MappingNotify** events to all clients whenever a client calls the following:

XSetModifierMapping which specifies the keycodes to be used as modifiers. The status reply must be **MappingSuccess**.

XChangeKeyboardMapping which defines the symbols for the keycodes to be changed.

XSetPointerMapping which sets the mapping of the pointer. The status reply must be **MappingSuccess**.

The following members of the **XMappingEvent** structure are associated with this event:

The *window* member which is not used in this event, but can be used with certain toolkits.

The *request* member which specifies the mapping change. It can be set to one of the following:

- **MappingModifier** if the specified keycodes are used as modifiers.
- **MappingKeyboard** if the keyboard mapping is changed.
- **MappingPointer** if the pointer button mapping is set.

The *first_keycode* member which specifies the first number in the range of altered keyboards. This is used only if *request* is **MappingKeyboard**.

The *count* member which specifies the last number in the range of altered keyboards. This is used only if *request* is **MappingKeyboard**.

To update the client application's knowledge of the keyboard, use **XRefreshKeyboardMapping**. See Chapter 2, "X-Windows Xlib Functions."

¦ Copyright IBM Corp. 1985, 1991

1.11.15.8 Processing ReparentNotify Events

The X Server reports **ReparentNotify** events to clients requesting information when the parent of a window is changed. It generates this event when a client application calls **XReparentWindow** and the window is actually reparented.

To receive a **ReparentNotify** event do one of the following:

Pass the window ID of the old or the new parent window an **SubStructureNotifyMask** as the *event_mask* argument to **XSelectInput**.

Pass the window ID and **StructureNotifyMask** as the *event_mask* argument to **XSelectInput**.

The following members of the **XReparentEvent** structure are associated with this event:

The *event* member which specifies the window ID of the window on which the **ReparentNotify** event was generated. This is also the window on which **XSelectInput** was selected. The X Server uses this window to report the event.

The *window* member which is the window ID of the window that was reparented.

The *parent* member which is the window ID of the new parent window.

The *x* and *y* members which are the coordinates that define the upper-left outer corner of the reparented window. These coordinates are relative to the origin of the new parent window.

The *override_redirect* member which is set to the value specified for the *override_redirect* member of the **XSetWindowAttributes** data structure when the window was created or when the window attributes were changed. This variable can be **True** or **False**. If *override_redirect* is **True**, client applications normally should ignore this event.

¦ Copyright IBM Corp. 1985, 1991

1.11.15.9 Processing UnmapNotify Events

The X Server reports **UnmapNotify** events to clients when requests are made to unmap windows. If a client application changes the window from mapped to unmapped with **XUnmapWindow** or **XUnmapSubwindows**, the X Server generates this request.

To receive **UnmapNotify** events, do one of the following:

Pass the window ID and **StructureNotifyMask** as the *event_mask* argument to **XSelectInput**.

Pass the window ID of the parent window and **SubstructureNotifyMask** as the *event_mask* argument to **XSelectInput**.

The members of the **XUnmapEvent** structure associated with this event include the following:

The *event* member which is the window ID of the window on which the **UnmapNotify** event was generated. This is also the window on which **XSelectInput** was initiated. The X Server uses this window to report the event.

The *window* member which is the window ID of the window that was unmapped.

The *from_configure* member which can be **True** or **False**. If the event was generated as a result of resizing the parent window when the window itself had a *win_gravity* of **UnmapGravity**, then *from_configure* is **True**.

¦ Copyright IBM Corp. 1985, 1991

1.11.15.10 Processing VisibilityNotify Events

The X Server reports **VisibilityNotify** events to clients requesting any change in the visibility of the window specified. (A region of a window is visible if it can be seen on the screen.) The X Server generates this event when the visibility of the window changes state. However, **VisibilityNotify** events are not generated for **InputOnly** windows or subwindows.

All **VisibilityNotify** events caused by a hierarchy change are generated after any hierarchy event (**UnmapNotify**, **MapNotify**, **ConfigureNotify**, **GravityNotify**, **CirculateNotify**) caused by that change. Any **VisibilityNotify** event on a window is generated before any **Expose** events on that window, but it is not required that all **VisibilityNotify** events on all windows are generated before all **Expose** events on all windows. The ordering of **VisibilityNotify** events with respect to **FocusOut**, **EnterNotify**, and **LeaveNotify** events is not constrained by the X protocol.

To receive **VisibilityNotify** events, pass the window ID of the window and **VisibilityChangeMask** as the *event_mask* argument to **XSelectInput**.

The members of the **XVisibilityEvent** structure associated with this event include the following:

The *window* member which is set to the window ID of the window whose visibility state changes.

The *state* member which is the visibility of the window. This can be set to **VisibilityUnobscured**, **VisibilityPartiallyObscured**, or **VisibilityFullyObscured**.

The X Server ignores the subwindows in determining the visibility state of the window and processes **VisibilityNotify** events according to the following:

When the window changes state from partially or fully obscured, or no viewable to viewable and completely unobscured, the X Server generates the event with **VisibilityUnobscured** as the *state* member of the **XVisibilityEvent** structure.

When the window changes state from viewable and completely unobscured or from not viewable to viewable and partially obscured, the X Server generates the event with **VisibilityPartiallyObscured** as the *state* member of the **XVisibilityEvent** structure.

When the window changes state from viewable and completely unobscured or viewable and partially obscured, or from not viewable to viewable and fully obscured, the X Server generates the event with **VisibilityFullyObscured** as the *state* member of the **XVisibilityEvent** structure.

¦ Copyright IBM Corp. 1985, 1991

1.11.16 Processing Structure Control Events

Structure control events are generated when clients have structure control enabled. These events, which are generally used only by window managers, include **CirculateRequest**, **ConfigureRequest**, **MapRequest**, and **ResizeRequest**.

Subtopics
1.11.16.1 Processing CirculateRequest Events
1.11.16.2 Processing ConfigureRequest Events
1.11.16.3 Processing MapRequest Events
1.11.16.4 Processing ResizeRequest Events

¦ Copyright IBM Corp. 1985, 1991

1.11.16.1 Processing CirculateRequest Events

The X Server reports **CirculateRequest** events to clients requiring information when another client initiates a window request on a specified parent window. The X Server generates this event type when a client initiates a circulate window request on a parent window and the window needs to be restacked. The client can initiate a circulate window request with **XCirculateSubwindows**, **XCirculateSubwindowsUp**, or **XCirculateSubwindowsDown**.

To receive a **CirculateRequest** event, pass the window ID of the parent window and **SubstructureRedirectMask** as the *event_mask* argument to **XSelectInput**. In the future, a circulate window request for any of the children of the specified window will not be executed and the position of the window in the stack is not changed. Instead, this client will receive a **CirculateRequest** event. Once this client has received the event, the client can perform **CirculateSubwindows** on the requesting client's window, or simply ignore the request, or possibly kill the requesting client. For example, suppose a client application calls **XCirculateSubwindowsUp** to raise a specified window to the top of the stack. If you had selected **SubstructureRedirectMask** on the parent window, the X Server reports a **CirculateRequest** event and does not raise the specified window to the top of the stack.

The members of the **XCirculateRequestEvent** structure associated with this event include the following:

The *parent* member which is the window ID of the parent window.

The *window* member which is the window ID of the window to be restacked.

The *place* member which is the new position in the stacking order. The new position can be **PlaceOnTop** or **PlaceOnBottom**.

- If *place* is **PlaceOnTop**, the window will be placed on top of all siblings.

- If *place* is **PlaceOnBottom**, the window will be placed below all siblings.

¦ Copyright IBM Corp. 1985, 1991

1.11.16.2 Processing ConfigureRequest Events

The X Server reports **ConfigureRequest** events to clients when another client initiates a configure request on a specified window. The configure window request attempts to reconfigure the window size, position, border, or stacking order. A **ConfigureRequest** event is generated and the client's attempt to reconfigure the window fails. A client can initiate the configure window request with one of the following:

XConfigureWindow	**XLowerWindow**	**XRaiseWindow**
XMapRaised	**XMoveResizeWindow**	**XMoveWindow**
XResizeWindow	**XRestackWindows**	**XSetWindowBorderWidth**

To receive a **ConfigureRequest** event, pass the window ID of the parent window and **SubstructureRedirectMask** as the *event_mask* argument to **XSelectInput**. For example, a configure request, **XLowerWindow**, is initiated to lower a window. The **SubstructureRedirectMask** is the event mask on the parent window and the *override_redirect* member of the **XSetWindowAttributes** structure associated with the specified window is set to **False**; then, the X Server reports a **ConfigureRequest** event, but does not lower the specified window.

The members of the **XConfigureRequestEvent** structure associated with this event include the following:

The *parent* member which is the window ID of the parent window.

The *window* member which is the window ID of the window to be reconfigured.

The *x* and *y* members which indicate the requested upper-left outside corner of the reconfigured window. These coordinates are relative to the origin of the parent window.

The *width* and *height* members which indicate the requested size of the reconfigured window, excluding the border. These dimensions are always a nonzero value.

The *border_width* member which is the requested width of the border (pixels) of the reconfigured window.

The *above* member which is the window ID of the sibling window. This position is for stacking operations.

- If *above* is **None**, the reconfigured window is placed on the bottom of the stack with respect to sibling windows.

- If *above* is the ID of a sibling window, the reconfigured window is placed on top of the sibling window.

The *detail* member which indicates the notify detail. This member can be set to **Below**, **TopIf**, **BottomIf**, or **Opposite**. If *detail* is not indicated in the request, it is set to **Above**

The *value_mask* member which indicates what components to specify in the request. This member and the corresponding values are reported as given in the request.

¦ Copyright IBM Corp. 1985, 1991

1.11.16.3 Processing MapRequest Events

The X Server reports **MapRequest** events to clients requiring information about a request by another client to map or place windows. (A window is considered mapped when a map window request completes.) A **MapRequest** event is generated when a client attempts to map a window and fails. Clients can initiate map window requests with **XMapWindow**, **XMapRaised**, or **XMapSubwindows**.

To receive **MapRequest** events, pass the window ID of the parent window and **SubstructureRedirectMask** as the *event_mask* argument to **XSelectInput**. When another client attempts to map the window and the attempt fails, the X Server sends a **MapRequest** event. For example, a client application calls **XMapWindow** to map a window. **SubstructureRedirectMask** is indicated on the parent window and the *override_redirect* member of the **XSetWindowAttributes** structure associated with the window is **False**; therefore, the window is not mapped and the X Server sends a **MapRequest** event. This gives the window manager client the ability to control the placement of subwindows.

The members of the **XMapRequestEvent** structure associated with this event include the following:

The *parent* member which is the window ID of the parent window.

The *window* member which is the window ID of the window to be mapped.

¦ Copyright IBM Corp. 1985, 1991

1.11.16.4 Processing ResizeRequest Events

The X Server reports **ResizeRequest** events to clients requesting information when another client attempts to change the size of a window. A client can attempt to change the size of a window by calling **XConfigureWindow**, **XResizeWindow**, or **XMoveResizeWindow**.

To receive **ResizeRequest** events, pass a window ID and **ResizeRedirect** as the *event_mask* argument to **XSelectInput**. When a client attempts to change the window size of another client window and the attempt fails, the X Server sends a **ResizeRedirect** event.

The members of the **XResizeRequestEvent** structure associated with this event include the following:

The *window* member which is the window ID of the target window (the window another client attempted to change).

The *width* and *height* members which indicate the size of the window, excluding the border.

¦ Copyright IBM Corp. 1985, 1991

1.11.17 Processing Colormap State Notification Events

The X Server reports **ColormapNotify** events to clients requiring information when the colormap changes and when a colormap is installed or uninstalled. A client application can change the *colormap* member of the **XSetWindowAttributes** structure with **XChangeWindowAttributes** or **XFreeColormap**. A client application can install or uninstall the colormap with **XInstallColormap** or **XUninstallColormap**.

To receive a **ColormapNotify** event, pass the window ID and **ColormapChangeMask** as the *event_mask* argument to **XSelectInput**.

The members of the **XColormapEvent** structure associated with this event include the following:

The *window* member which is the window ID of the target window (the window whose colormap is changed, installed, or uninstalled).

- For a colormap changed by **XChangeWindowAttributes**, the *colormap* member is the colormap resource ID of the colormap associated with the window.

- For a colormap changed by **XFreeColormap**, the *colormap* member is **None**.

The *new* member which indicates whether the colormap for the specified window was changed or installed or uninstalled. This variable can be one of the following:

- **True** if the colormap was changed.

- **False** if the colormap was installed or uninstalled.

The *state* member which indicates if the colormap is installed or uninstalled. This variable can be **ColormapInstalled** or **ColormapUninstalled**.

¦ Copyright IBM Corp. 1985, 1991

1.11.18 Processing Client Communication Events

Processing client communication events is done with different events and different requirements. The X Server generates **ClientMessage** events when a client calls **XSendEvent**. It reports **PropertyNotify** events to clients requesting information about property changes to a window. The X Server reports **SelectionClear** and **SelectionRequest** events to the current owner of a selection. It also reports **SelectionNotify** events, but only in response to an **XConvertSelection** request.

Subtopics
1.11.18.1 Processing ClientMessage Events
1.11.18.2 Processing PropertyNotify Events
1.11.18.3 Processing SelectionClear Events
1.11.18.4 Processing SelectionRequest Events
1.11.18.5 Processing SelectionNotify Events

¦ Copyright IBM Corp. 1985, 1991

1.11.18.1 Processing ClientMessage Events

The X Server generates **ClientMessage** events when a client calls the **XSendEvent** function. The **XSendEvent** function identifies the destination window, determines which clients should receive the specified events, and ignores any active grabs. See Chapter 2, "X-Windows Xlib Functions" for information about this function.

The members of the **XClientMessageEvent** structure associated with the **ClientMessage** event include the following:

The *window* member which is the window ID of the window to which the event was sent.

The *message_type* member which is an atom that indicates how the data is to be interpreted by the receiving client. This member is not interpreted by the X Server.

The *format* member which specifies if the data should be viewed as a list of bytes, shorts, or longs. This member should be set to 8-bits, 16-bits, or 32-bits.

The *data* member which is a union that contains the list of b (bytes), s (shorts), and l (longs). These members represent data of twenty 8-bit values, ten 16-bit values, and five 32-bit values. This member is not interpreted by the X Server. (Some message types may not use all these values.)

¦ Copyright IBM Corp. 1985, 1991

1.11.18.2 Processing PropertyNotify Events

The X Server reports **PropertyNotify** events to clients requesting information about property changes for a specified window. (A property consists of an atom name, an atom type, a data format, and some property data.) The X Server generates this event when a client application calls **XChangeProperty**, **XDeleteProperty**, **XRotateWindowProperties**, or **XGetProperty**.

To receive a **PropertyNotify** event, pass the window ID and **PropertyChangeMask** as the *event_mask* argument to **XSelectInput**.

The members of the **XPropertyEvent** structure associated with this event include the following:

The *window* member which is the window ID of the window whose property is changed.

The *atom* member which is the atom of the property that is changed or requested.

The *time* member which is the server time when the property is changed.

The *state* member which indicates if the property is changed to a new value or deleted. It can be **PropertyNewValue** or **PropertyDelete**.

¦ Copyright IBM Corp. 1985, 1991

1.11.18.3 Processing SelectionClear Events

The X Server reports **SelectionClear** events to the current owner of a selection. This event is generated on the window losing ownership of the selection to a new owner. The function **XSetSelectionOwner** sets the selection owner. See Chapter 2, "X-Windows Xlib Functions" for information about this function. See "Using Window Selections" in topic 1.6.5 for information on selection.

The members of the **XSelectionClearEvent** structure associated with this event include the following:

The *window* member which is the ID of the window losing ownership of the selection.

The *selection* member which is the selection atom.

The *time* member which is the last change time recorded for the selection.

The *owner* member which is the window specified by the current owner in the **XSetSelectionOwner** call.

¦ Copyright IBM Corp. 1985, 1991

1.11.18.4 Processing SelectionRequest Events

The X Server reports **SelectionRequest** events to the owner of a selection when a client requests a selection conversion with **XConvertSelection** and the selection specified is owned by a window.

The members of the **XSelectionRequestEvent** structure associated with this event include the following:

The *owner* member which is the window ID of the window owning the selection. This is the window specified by the current owner in **XSetSelectionOwner**.

The *requestor* member which is the window ID of the window requesting the selection.

The *selection* member which is the atom that indicates the selection. For example, PRIMARY indicates the primary selection. See "Using Properties and Atoms" in topic 1.6.4.

The *target* member which is the atom that indicates the type requested.

The *property* member which can be an atom, the property, or **None**.

The *time* member which is the time, either in a timestamp (milliseconds) or in **CurrentTime**, taken from the **XConvertSelection** request.

The client that owns the selection should do the following:

Convert the selection based on the atom contained in the *target* member.

Store the result as that property on the *requestor* window; then send a **SelectionNotify** event to the *requestor* with **XSendEvent** with an empty *event_mask*. The event should be sent to the client of the *requestor* window if a property was specified or the *property* member was set.

Send a **SelectionNotify** event with the property set to **None** if the selection cannot be converted as requested.

¦ Copyright IBM Corp. 1985, 1991

1.11.18.5 Processing SelectionNotify Events

The X Server sends **SelectionNotify** events in response to a **XConvertSelection** request. If the selection has an owner, the **SelectionNotify** event should be generated by the owner with **XSendEvent**. If the selection has been converted and stored as a property or a selection conversion could not be performed, the **SelectionNotify** event should be sent to the *requestor* window.

If **None** is specified as the property, the selection owner should choose a property name, store the results as that property on the requestor window, and then send a **SelectionNotify** event.

The members of the **XSelectionEvent** structure associated with this event include the following:

The *requestor* member which is the window ID of the window associated with the requestor of the selection.

The *selection* member which is the atom that indicates the kind of selection. For example, PRIMARY is used for the primary selection. See "Using Properties and Atoms" in topic 1.6.4.

The *target* member which is the atom that indicates the type requested. For example, PIXMAP is used for a pixmap.

The *property* member which is the atom that indicates the property the result is stored on.

The *time* member which is the time when the conversion took place. This can be a timestamp (in milleseconds) or **CurrentTime**.

¦ Copyright IBM Corp. 1985, 1991

1.11.19 Selecting Events

There are two ways to select the events reported to your client application. First, set the *event_mask* member of the **XSetWindowAttributes** structure when you use **XCreateWindow** and **XChangeWindowAttributes**. Second, use **XSelectInput**. See Chapter 2, "X-Windows Xlib Functions" for information about these functions.

Events are reported relative to a window. If a window requests an event, it usually propagates to the closest ancestor that does not request an event unless the *do_not_propagate* mask prohibits propagation.

A call to **XSelectInput** overrides any previous call to the same function for the same window from the same client, but not from other clients. Multiple clients can select for the same events on the same window with the following restrictions:

Multiple clients can select events on the same window because th event masks are disjointed. After the X Server generates an event, it reports the event to all interested clients.

Only one client at a time can select **CirculateRequest**, **ConfigureRequest**, or **MapRequest** events, which are associated with the **SubstructureRedirectMask** event mask.

Only one client at a time can select a **ResizeRequest** event, which is associated with the **ResizeRedirectMask** event mask.

Only one client at a time can select a **ButtonPress** event, which is associated with the **ButtonPressMask** event mask.

The server reports the event to all interested clients.

Subtopics
1.11.19.1 Related Functions

¦ Copyright IBM Corp. 1985, 1991

1.11.19.1 Related Functions

XSelectInpu

¦ Copyright IBM Corp. 1985, 1991

1.11.20 Handling the Output Buffer and the Event Queue

Under some circumstances, many event functions flush the output buffer. (The ***output buffer*** is an area used by the **Xlib** library to store requests.) Some event functions flush the input queue if the function would block or not return an event. All requests residing in the output buffer that have not yet been sent are transmitted to the X Server. Conversely, these functions differ in the additional tasks they might perform. For example, **XSync** not only flushes the output buffer, but it can also discard all events in the event queue.

Subtopics
1.11.20.1 Related Functions

¦ Copyright IBM Corp. 1985, 1991

1.11.20.1 Related Functions

XFlush

XEventsQueued

XNextEvent

XWindowEvent

XMaskEvent

XCheckTypedEvent

XSync

XPending

XPeekEvent

XCheckWindowEvent

XCheckMaskEvent

XCheckTypedWindowEvent

¦ Copyright IBM Corp. 1985, 1991

1.11.21 Selecting Events Using a Predicate Procedure

The functions **XIfEvent**, **XCheckIfEvent**, and **XPeekIfEvent** require a predicate procedure that determines if the event matches the event specified in the corresponding function. The predicate procedure must decide only if the event is useful and must not call **Xlib** functions. This predicate procedure is called from within the event routine, which must be locked so that the event queue is consistent in a multi-threaded environment.

The predicate procedure for these functions is defined as:

```
Bool (*predicate)(display, event, args);
Display *display;
XEvent *event;
char *args;
```

display Specifies the connection to the X Server.

event Specifies a pointer to the structure **XEvent**.

args Specifies the arguments passed in from the **XIfEvent**, **XCheckIfEvent**, or **XPeekIfEvent** function.

The predicate procedure is called once for each event in the queue until it finds a match between the event in the queue and the event specified by the corresponding function. This procedure returns **True** if it finds a match. It returns **False** if it does not find a match.

Subtopics
1.11.21.1 Related Functions

¦ Copyright IBM Corp. 1985, 1991

1.11.21.1 Related Functions

XIfEvent

XCheckIfEvent

XPeekIfEvent

¦ Copyright IBM Corp. 1985, 1991

1.11.22 Using the Default Error Handler

Two default error handlers reside in the library: **XSetIOErrorHandler** for typically fatal conditions (for example, the connection to the display fails due to a system crash), and **XSetErrorHandler** for error events from the X Server. These error handlers can be changed to user-supplied routines as often as necessary. To reinvoke either error handler, pass a NULL pointer. The default is to print an explanatory message and exit.

The **XErrorEvent** data structure is defined as:

```
typedef struct {

   int type;
   Display *display;          /* display the event was read from       */

   int serial;                /* serial number of failed request       */
   char error_code;           /* error code of failed request          */
   char request_code;         /* major op-code of failed request       */
   char minor_code;           /* minor op-code of failed request       */
   XID resourceid;            /* resource id                           */

} XErrorEvent;
```

The *serial* member is the number of requests starting with the one sent over the network connection when it was opened. This value is given to the request sequence number immediately after the failing call was made.

The *request_code* member is a protocol representation of the procedure that failed. The *request_code* members are defined in **<X11/X.h>**.

The error codes generated by X-Windows are in Appendix F, "Error Codes" in topic F.0.

Subtopics
1.11.22.1 Related Functions

¦ Copyright IBM Corp. 1985, 1991

1.11.22.1 Related Functions

XSetErrorHandle

XGetErrorTex

XGetErrorDatabaseTex

XDisplayNam

XSetIOErrorHandle

¦ Copyright IBM Corp. 1985, 1991

1.12 Using Predefined Property Functions

There are a number of predefined properties for common information usually associated with windows. The atoms for these properties are in the file **<X11/Xatom.h>**.

Xlib functions perform predefined property operations used in communicating with window managers.

Subtopics

¦ Copyright IBM Corp. 1985, 1991

1.12.1 Communicating with Window Managers

Clients require certain properties and functions to communicate with window managers effectively. Some of these properties have complex structures. For example, the data in a single property on the server has to be of the same format (8-bit, 16-bit, or 32-bit), and the structures of property types are not necessarily uniform. Therefore, **Xlib** provides set and get functions.

These functions define, but do not enforce, minimal policy among window managers. It is encouraged that standard properties be used in creating window managers. However, additional properties may be defined for new window managers.

The Set function redefines all the fields in the property even though a new value may have been specified for some of the fields only.

In addition, the **Xlib** functions **XSetStandardProperties** and **XChangeProperty** set all, or portions, of simple properties for less complex structures. **XGetWindowProperty** retrieves the values set with **XChangeProperty**.

To work well with most window managers, an application should specify the name of the application, the name string for the icon, the command used to invoke the application, and the size and window manager hints.

X-Windows does not set defaults for the properties. The window manager determines the defaults which are based on the presence or absence of certain properties. All the properties are considered ***hints*** to a window manager. When implementing these hints, the window manager decides whether or not to use them.

Predefined properties include the following :

Name	**Type**	**Format**	**Description**
WM_NAME	STRING	8	Application name.
WM_ICON_NAME	STRING	8	Icon name.
WM_NORMAL_HINTS	WM_SIZE_HINTS	32	Size hints for a window in its normal state. (**XSizeHints**)
WM_ZOOM_HINTS	WM_SIZE_HINTS	32	Size hints for a zoomed window. (**XSizeHints**)
WM_HINTS	WM_HINTS	32	Additional hints set by client for the window manager. (**XWMHints**)
WM_COMMAND	STRING	8	The command and arguments, separated by ASCII 0s, used to

¦ Copyright IBM Corp. 1985, 1991

			invoke the application.
WM_ICON_SIZE	WM_ICON_SIZE	32	The window manager can set this property on the root window to specify the icon sizes it supports. (**XIconSize**)
WM_CLASS	STRING	32	Set by application programs to allow window and session managers to obtain the application resources from the resource database.
WM_TRANSIENT_FOR	WINDOW	32	Set by application programs to indicate to the window manager that a transient top-level window, such as a dialog box, is not really a full-fledged window.

The atom names stored in **<X11/Xatom.h>** are named **XA**_*PROPERTY_NAME*.

Subtopics
1.12.1.1 Setting Standard Properties
1.12.1.2 Related Functions

¦ Copyright IBM Corp. 1985, 1991

1.12.1.1 Setting Standard Properties

Xlib provides **XSetStandardProperties** to set all, or portions of following properties: WM_NAME, WM_ICON_NAME, WM_HINTS, WM_COMMAND, and WM_NORMAL_HINTS.

¦ Copyright IBM Corp. 1985, 1991

1.12.1.2 Related Functions

XSetStandardPropertie

¦ Copyright IBM Corp. 1985, 1991

1.12.2 Setting and Getting Window Names

Xlib provides functions to set and read the name of a window. **XStoreName** assigns a name to a window. This window name is usually displayed by the window manager in a titlebar. **XFetchName** gets the name of a window. It obtains a window name if the WM_NAME property is set.

Subtopics
1.12.2.1 Related Functions

¦ Copyright IBM Corp. 1985, 1991

1.12.2.1 Related Functions

XStoreNam

XFetchNam

¦ Copyright IBM Corp. 1985, 1991

1.12.3 Setting and Getting Icon Names

Xlib provides functions to set or get the name displayed in an icon window. **XSetIconName** sets the name displayed in the icon window. **XGetIconName** gets the name set in the icon window by **XSetIconName**.

Subtopics
1.12.3.1 Related Functions

¦ Copyright IBM Corp. 1985, 1991

1.12.3.1 Related Functions

XSetIconNam

XGetIconNam

¦ Copyright IBM Corp. 1985, 1991

1.12.4 Setting and Getting Window Manager Hints

Xlib provides functions that set or get window manager hints. All the properties of a window are considered hints to a window manager. The window manager decides whether to implement these hints. (See "Communicating with Window Managers" in topic 1.12.1.) Functions that set or get window manager hints are:

XSetWMHints which sets the window manager hints that include icon information, initial state of the window, and keyboard input.

XGetWMHints which reads the value of the window manager hints atom.

When setting and reading the WM_HINTS property, both functions use the following **XWMHints** data structure:

```
typedef struct {
    long flags;                 /* marks which fields in this          */
                                   structure are defined
    Bool input;                 /* type of input, for example,         */
                                   keyboard input
    int initial_state;          /* initial state of the application    */
    Pixmap icon_pixmap;         /* pixmap to be used as icon           */
    Window icon_window;         /* window to be used as icon           */
    int icon_x, icon_y;         /* initial position of icon            */
    Pixmap icon_mask;           /* pixmap to be used as mask for       */
                                   icon_pixmap
    XID window_group;           /* ID of related window group          */
    unsigned int messages;      /* messages                            */
} XWMHints;
```

The members of the **XWMHINTS** data structure include the following:

The *flags* variable which lists fields defined in the **XWMHINTS** data structure. The values for this variable are:

```
#define   InputHint                 (1L<<0)
#define   StateHint                 (1L<<1)
#define   IconPixmapHint            (1L<<2)
#define   IconWindowHint            (1L<<3)
#define   IconPositionHint          (1L<<4)
#define   IconMaskHint              (1L<<5)
#define   WindowGroupHint           (1L<<6)
#define   MessageHint               (1L<<7)
#define   AllHints
```

The *input* field which communicates the input focus model to the window manager used by the application.

- If *input* is **True**, the application accepts input, but never explicitly sets focus to the driven focus of the subwindows. (These applications use the push-model of focus management.)

- If *input* is **True**, the application sets input focus to the subwindows only when the focus is given to the top-level window by a window manager.

  If *input* is **True**, pull-model window managers should make it

¦ Copyright IBM Corp. 1985, 1991

possible for push-model applications to get input by setting input focus to the top-level windows of applications.

- If *input* is **False**, the application manages its input focus by explicitly setting focus to one of the subwindows whenever keyboard input is requested. (These applications use the pull-model of focus management.)

- If *input* is **False**, the application is not required to accept keyboard input.

Resetting input focus to **PointerRoot** sets input focus to one of its subwindows if the *input* is **False**.

Push-model window managers should make sure that pull-model applications do not break them.

The *initial_state* flag which defines the state of the application. The values for this flag are:

```
#define   WithdrawnState          0
#define   NormalState             1
#define   IconicState             3
```

Most applications set the *initial_state* to **NormalState**. If an application is seldom used, the window manager sets the *initial_state* to **InactiveState**.

The *icon_mask* which specifies what pixels of the *icon_pixmap* should be used as the icon. The *icon_mask* allows for nonrectangular pixmaps.

The *window_group* which specifies if this window belongs to a group of other windows. For example, if a single application manipulates multiple children of the root window, this variable provides the window manager with enough information to iconify all of the windows instead of only one window.

Subtopics
1.12.4.1 Related Functions

¦ Copyright IBM Corp. 1985, 1991

1.12.4.1 Related Functions

XSetWMHint

XGetWMHint

¦ Copyright IBM Corp. 1985, 1991

1.12.5 Setting and Getting Window Manager Sizing Hints

Xlib provides functions that set or get window sizing hints. All the properties of a window are considered hints to a window manager. The window manager decides whether to implement these hints. (See "Communicating with Window Managers" in topic 1.12.1) Functions that set or get window sizing hints are:

XSetNormalHints which sets the size hints for a window in its normal state.

XGetNormalHints which returns the size hints for a window in its normal state.

XSetZoomHints which sets the value of the zoom hints atom.

XGetZoomHints which reads the value of the zoom hints atom.

XSetSizeHints which sets the value of a property of type WM_SIZE_HINTS.

XGetSizeHints which reads the value of a property of type WM_SIZE_HINTS.

These functions use the following **XSizeHints** data structure:

```
typedef struct {
   long flags;                    /* marks the defined fields        */
   int x, y;
   int width, height;
   int min_width, min_height;
   int max_width, max_height;
   int width_inc, height_inc;
   struct {
      int x;                      /* numerator                       */
      int y;                      /* denominator                     */
   } min_aspect, max_aspect;
   int base_width, base_height;

} XSizeHints;
```

The members in the **XSizeHints** data structure include the following:

The *flags* field which determines how the position and size of the window is set. The values for this field are:

```
#define   USPosition              (1L<<0)
#define   USSize                  (1L<<1)
#define   PPosition               (1L<<2)
#define   PSize                   (1L<<3)
#define   PMinSize                (1L<<4)
#define   PMaxSize                (1L<<5)
#define   PResizeInc              (1L<<6)
#define   PAspect                 (1L<<7)
#define   PBaseSize               (1L<<8)
#define   PAllHints               user specified
```

- If *flags* is set to **USPosition** and **USSize**, you specify the position and the size.

¦ Copyright IBM Corp. 1985, 1991

- If *flags* is set to **PPosition** and **PSize**, you specify that the program will indicate the position and size of the window.

The *x, y, width,* and *height* members which describe a position and size for the window.

The *min_width* and *min_height* members which specify the minimum size of the window for the application.

The *max_width* and *max_height* members which specify the maximum size of the window.

The *width_inc* and *height_inc* members which define an arithmetic progression of sizes, from minimum size to maximum size, for the window resize requests.

The *min_aspect* and *max_aspect* members which are expressed as ratios of *x* and *y*. These members specify the range of aspect ratios the application prefers.

The *base_width* and *base_height* members which, with the *width_inc* and *height_inc* members, define an arithmetic progression of the preferred window width and height.

```
   width   =  base_width   +   (  i * width_inc  )
  height   =  base_height  +   (  j * height_inc )
```

Subtopics
1.12.5.1 Related Functions

¦ Copyright IBM Corp. 1985, 1991

1.12.5.1 Related Functions

XSetNormalHint

XGetNormalHint

XSetZoomHint

XGetZoomHint

XSetSizeHint

XGetSizeHint

¦ Copyright IBM Corp. 1985, 1991

1.12.6 Setting and Getting Icon Sizing Hints

Applications can cooperate with window managers by providing icons in sizes supported by the window manager. To communicate the supported icon sizes to the applications, a window manager should set the icon size property on the root window. To determine what icon sizes a window manager supports, applications should read the WM_ICON_SIZE property from the root window. The following functions set or read the WM_ICON_SIZE property in the window manager:

XSetIconSizes sets the value of the icon size atom.

XGetIconSizes returns the value of the icon size atom.

Both functions use the following **XIconSize** data structure:

```
typedef struct {

    int min_width, min_height;
    int max_width, max_height;
    int width_inc, height_inc;

} XIconSize;
```

The members in the **XIconSize** data structure are:

The *min_width* and *min_height* members which specify the minimum icon size.

The *max_width* and *max_height* members which specify the maximum icon size.

The *width_inc* and *height_inc* members which define an arithmetic progression of sizes, from minimum to maximum, that represent the supported icon sizes.

Subtopics
1.12.6.1 Related Functions

¦ Copyright IBM Corp. 1985, 1991

1.12.6.1 Related Functions

XSetIconSize

XGetIconSize

¦ Copyright IBM Corp. 1985, 1991

1.12.7 Setting and Getting the Class of a Window

Xlib provides functions that set and get the class of a window with the WM_CLASS property. This property is set by applications to allow the window and session managers to obtain the client's resources from the resource database. The following functions set and read the WM_CLASS property:

XSetClassHint sets the class of a window.

XGetClassHint gets the class of a window.

These functions use the following **XClassHint** data structure:

```
typedef struct{

    char *res_name;
    char *res_class;

} XClassHint;
```

The members of the **XClassHint** data structure are:

The *res_name* member which contains the application name.

The *res_class* member which contains the application class.

The name set in WM_CLASS can be different than the name set in WM_NAME.

WM_NAME specifies the name that is displayed in the titlebar. Thi name may be temporary information such as the name of a file currently in the buffer of an editor.

WM_CLASS is the application name used when retrieving th application's resources from the resource database.

Subtopics
1.12.7.1 Related Functions

¦ Copyright IBM Corp. 1985, 1991

1.12.7.1 Related Functions

XSetClassHin

XGetClassHin

¦ Copyright IBM Corp. 1985, 1991

1.12.8 Setting and Getting the Transient Property

An application can indicate to the window manager that a transient top-level window (for example, a dialog box) is not really a full-fledged window. Rather, the top-level window is operating on behalf of another window or is transient for another window. To do this, the application sets the WM_TRANSIENT_FOR property of the dialog box to be the window handle of the main window.

Some window managers use this information to unmap dialog boxes of an application, for example, when the main application window is iconified.

Subtopics
1.12.8.1 Related Functions

¦ Copyright IBM Corp. 1985, 1991

1.12.8.1 Related Functions

XSetTransientForHin

XGetTransientForHin

¦ Copyright IBM Corp. 1985, 1991

1.13 Using the Resource Manager

The resource manager is a database manager. In most database systems, you perform a query using an imprecise specification to receive a set of records. The resource manager, however, allows you to specify a large set of values with an imprecise specification, to query the database with a precise specification, and to return only a single value. This function should be used by applications that need to know what the user prefers for colors, fonts, and other resources.

For example, someone using your application may want to specify that all windows should have a blue background but that all mail reading windows should have a red background. Presuming that all applications use the resource manager, a user can define this information using only two lines of specification. Your personal resource database usually is stored in a file and is loaded onto a server property when you login. This database is retrieved automatically by **Xlib** when a connection is opened.

As an example of how the resource manager works, consider a mail reading application called **xmh**. Assume that it is designed in such a manner that it uses a complex window hierarchy, all the way down to individual command buttons that can be actual small subwindows. These are often called ***objects***. These user interface objects can be composed of other objects. Each user interface object can be assigned a name and a class. Fully qualified names or classes can have arbitrary numbers of component names. This naming convention generally reflects the structure of the application as composed of these objects, starting with the application itself.

For example, the **xmh** mail program has a name **xmh** and is one of a class of *Mail* programs. By convention, the first character of class components is capitalized while the first letter of name components is in lowercase. Each name and class also has an attribute, for example, *foreground* of font. If each window is properly assigned a name and a class, it becomes easy for the user to specify attributes of any portion of the application.

At the top level, the application might consist of a ***paned*** window-- a window divided into several sections--named *toc*. One pane of the window is a button box window named *buttons* filled with command buttons. One of these command buttons is used to retrieve (include) new mail and has the name **include.** This window has a fully qualified name **xmh.toc.buttons.include** and a fully qualified class **Xmh.VPaned.Box.Command**. Its class is the class of its parent window, **Xmh.VPaned.Box**, followed by its particular class, **Command**. The fully qualified name of a resource is the attribute name appended to the fully qualified name of the object, and the fully qualified class is its class appended to the class of the object.

The **include** button requires the following resources:

```
Title strin
Fon
Foreground and background colo
Foreground and background color for its active stat
```

Each resource that this button needs is considered an attribute of the button and, as such, has a name and a class. For example, the foreground color for the button in its active state might be **activeForeground** and the class could be **Foreground.**

When an application searches for a resource (for example, a color), it

¦ Copyright IBM Corp. 1985, 1991

passes the complete name and class of the resource to a lookup routine. Then, the resource manager returns the resource value and the representation type.

The resource manager allows applications to store resources by an incomplete specification of name, class, and representation type, as well as to retrieve them given a fully qualified name and class.

Most uses of the resource manager involve defining names, classes, and representation types as string constants. However, always referring to strings in the resource manager can be slow, because it is used so much by some toolkits. To solve this, a shorthand name for a string is used in place of the full name of the string when the resource manager is being used in many of the resource manager functions. Simple comparisons can be performed rather than string comparisons. The short name for a string is ***quark***. The quark type is **XrmQuark**. You may want to allocate a quark that has no string equivalent on some occasions. (A quark is to a string what an atom is to a property name in the server, but the use of a quark is local to your application.)

Names, classes, and representation types are defined as **XrmQuarks**:

```
typedef int XrmQuark, *XrmQuarkList;
typedef XrmQuark XrmName;
typedef XrmQuark XrmClass;
typedef XrmQuark XrmRepresentation;
```

Lists are represented as null-terminated arrays of quarks. The size of the array must be large enough for the number of components used.

```
typedef XrmQuarkList XrmNameList;
typedef XrmQuarkList XrmClassList;
```

Subtopics

¦ Copyright IBM Corp. 1985, 1991

1.13.1 Resource Manager Matching Rules

The algorithm for determining which resource name matches a given query is the heart of the database. Resources are stored with only partially specified names and classes, using pattern matching constructs. An asterisk (*) is used to represent any number of intervening components, including none. A period (.) is used to separate immediately adjacent components. All queries fully specify the name and class of the resource needed. The lookup algorithm then searches the database for the name that matches the closest (is most specific) to this full name and class. The rules in order of precedence for a match are:

1. The attribute of the name and class must match. For example, queries for

   ```
   aixterm.scrollbar.background          (name)
   AIXTerm.Scrollbar.Background          (class)
   ```

   will not match the database entry

   ```
   aixterm.scrollbar:on
   ```

2. Database entries with a name or class prefixed by a period (.) are more specific than those prefixed by an asterisk (*). For example, **aixterm.geometry** is more specific than **aixterm*geometry**.

3. Names are more specific than classes. For example, ***scrollbar.background** is more specific than ***Scrollbar.Background**.

4. A name or class is more specific than an omission. For example, **Scrollbar*Background** is more specific than ***Background**.

5. Left components are more specific than right components. For example, **aixterm*background** is more specific than **scrollbar*background**.

6. If neither a period (.) nor an asterisk (*) is specified at the beginning, a period is implicit. For example, **aixterm.background** is identical to **.aixterm.background**.

As an example of these rules, assume the following user preference specification:

```
xmh*background:                    red
*command.font:                     8x13
*command.background:               blue
*Command.Foreground:               green
xmh.toc*Command.activeForeground:  black
```

A query for the name

xmh.toc.messagefunctions.include.activeForeground

and class

Xmh.VPaned.Box.Command.Foreground

would match

xmh.toc*Command.activeForeground

¦ Copyright IBM Corp. 1985, 1991

and return **black**. However, it also matches

***Command.Foreground**.

Using the precedence algorithm described above, the resource manager would return the value specified by **xmh.toc*Command.activeForeground**.

¦ Copyright IBM Corp. 1985, 1991

1.13.2 Basic Resource Manager Definitions

The definitions used by the resource manager are contained in the **<X11/Xresource.h>** header file. **Xlib** also uses the resource manager internally to allow for non-English language error messages.

Database values consist of a size, an address, and a representation type. The size is specified in bytes. The representation type is a way to store data tagged by some memory application-defined type, such as *font* or *color*. (Representation type is not the same as the C language data type or class.)

The **XrmValue** data structure contains:

```
typedef struct{

    unsigned int size;
    caddr_t addr;

}XrmValue, *XrmValuePtr;
```

A resource database is an opaque type used by the lookup routines.

```
typedef struct _  XrmHashBucketRec *XrmDatabase;
```

Subtopics
1.13.2.1 Related Functions

¦ Copyright IBM Corp. 1985, 1991

1.13.2.1 Related Functions

XrmInitialize

XrmUniqueQuark

Xpermalloc

XrmStringToQuark

XrmQuarkToString

XrmStringToQuarkList

XrmBindLoosely

XrmBindTightly

XrmStringToBindingQuarkList

XrmQPutResource

XrmPutStringResource

XrmQPutStringResource

XrmPutLineResource

¦ Copyright IBM Corp. 1985, 1991

1.13.3 Looking Up from a Resource Database

To retrieve a resource from a specified resource database, use **XrmGetResource** or **XrmQGetResource**. Both functions take a fully qualified name and class pair, a destination resource representation, and the address of a value (size and address pair). The value returns points into database memory that should not be modified.

Currently, the database frees or overwrites entries only on **XrmPutResource**, **XrmQPutResource**, or **XrmMergeDataBases**. A client that is not storing new values and is not merging the database should be safe using the address passed back at any time until it exits. If a resource is found, both **XrmGetResource** and **XrmQGetResource** return **True**.

Subtopics
1.13.3.1 Related Functions

¦ Copyright IBM Corp. 1985, 1991

1.13.3.1 Related Functions

XrmQGetSearchList

XrmQGetSearchResource

XrmMergeDatabases

XrmGetFileDatabase

XrmPutFileDatabase

XrmGetStringDatabase

XrmParseCommand

¦ Copyright IBM Corp. 1985, 1991

1.14 Using the Context Manager

The context manager provides a way of associating data with a window in a program. The context manager is local to your program because the data is not stored in the server on a property list. Any data in any number of pieces can be associated with a window, and each piece of data has a type associated with it. The context manager requires the window ID and type to store or retrieve data.

Essentially, the context manager can be viewed as a two-dimensional, sparse array: one dimension is subscripted by the window ID and the other by a context type field. Each entry in the array contains a pointer to the data.

Xlib provides context management functions to save and get data values, delete entries, and create a unique context type. The symbols used are in the **<X11/Xutil.h>** header file.

Subtopics
1.14.1 Related Functions

¦ Copyright IBM Corp. 1985, 1991

1.14.1 Related Functions

XSaveContex

XFindContex

XDeleteContex

XUniqueContex

¦ Copyright IBM Corp. 1985, 1991

2.0 Chapter 2. X-Windows Xlib Functions

Subtopics

¦ Copyright IBM Corp. 1985, 1991

2.1 CONTENTS

¦ Copyright IBM Corp. 1985, 1991

2.2 About This Chapter

This chapter describes the X-Windows functions.

Subtopics
2.2.1 What You Need to Know
2.2.2 How This Chapter Is Organized

¦ Copyright IBM Corp. 1985, 1991

2.2.1 What You Need to Know

In order to use this chapter, you should be familiar with, and be able to write programs in, the **C** programming language. For more information, see *C Language Guide and Reference*.

¦ Copyright IBM Corp. 1985, 1991

2.2.2 How This Chapter Is Organized

Subroutines -- X-Windows subroutines are listed according to the typ of function provided.

Subroutine Reference Information -- Each of the X-Windows progra interface subroutines is described. The subroutines are listed in alphabetical order.

If you are a first-time user, you should read and become familiar with the information in Chapter 1, "Using X-Windows," before attempting to use the subroutine reference information.

¦ Copyright IBM Corp. 1985, 1991

2.3 Subroutines

The following is a list of the X-Windows subroutines grouped according to the type of function provided.

Opening and Closing Display

XOpenDisplay- Open a display.
XNoOp- Execute a NoOperation protocol request.
XFree- Free in-memory data created by Xlib function.
XCloseDisplay- Close a display.

Creating and Destroying Window

XCreateWindow- Create unmapped subwindow.
XCreateSimpleWindow- Create unmapped InputOutput subwindow.
XDestroyWindow- Unmap and destroy window and all subwindows.
XDestroySubwindows- Destroy all subwindows of specified window.

Manipulating Window

XMapWindow- Map the specified window.
XMapRaised- Map and raise the specified window.
XMapSubwindows- Map all subwindows of the specified window.
XUnmapWindow- Unmap the specified window.
XUnmapSubwindows- Unmap all subwindows of the specified window.
XConfigureWindow- Configure the specified window.
XMoveWindow- Move the specified window.
XResizeWindow- Change the specified window's size.
XMoveResizeWindow- Change the specified window's size and location.
XSetWindowBorderWidth- Change the border width of the window.
XRaiseWindow- Raise the specified window.
XLowerWindow- Lower the specified window.
XCirculateSubwindows- Circulate a subwindow up or down.
XCirculateSubwindowsUp- Raise the lowest mapped child of window.
XCirculateSubwindowsDown- Lower the highest mapped child of window.
XRestackWindows- Restack a set of windows from top to bottom.

Changing Window Attribute

XChangeWindowAttributes- Change one or more window attributes.
XSetWindowBackground- Set window's background to specified pixel.
XSetWindowBackgroundPixmap- Set window's background to specified pixmap.
XSetWindowBorder- Change window's border to specified pixel.
XSetWindowBorderPixmap- Change window's border tile.
XTranslateCoordinates- Transform coordinates betweens windows.

Obtaining Window Informatio

XQueryTree- Obtain the IDs of the children and the parent windows.
XGetWindowAttributes- Get current attributes for specified window.
XGetGeometry- Get current geometry of specified drawable.
XQueryPointer- Get pointer coordinates and root window.

Properties and Atom

XInternAtom- Get an atom for the specified name.

¦ Copyright IBM Corp. 1985, 1991

XGetAtomName- Get a name for the specified atom ID.

Manipulating Window Propertie

XGetWindowProperty- Get atom type and property format for window.
XListProperties- Get the specified window's property list.
XChangeProperty- Change the property for specified window.
XRotateWindowProperties- Rotate properties in property array.
XDeleteProperty- Delete a property for the specified window.

Setting Window Selection

XSetSelectionOwner- Set the selection owner.
XGetSelectionOwner- Get the selection owner.
XConvertSelection- Convert a selection.

Manipulating Colormap

XCreateColormap- Create a colormap.
XCopyColormapAndFree- Create a new colormap from specified colormap.
XSetWindowColormap- Set the colormap of the specified window.
XFreeColormap- Free the specified colormap.
XQueryColor- Query the RGB value for a specified pixel.
XQueryColors- Query the RGB values for array of pixels.

Manipulating Color Cell

XAllocColor- Allocate a read-only color cell.
XAllocNamedColor- Allocate a read-only color cell by name.
XLookupColor- Look up colorname.
XAllocColorCells- Allocate read/write color cells.
XAllocColorPlanes- Allocate read/write color resources.
XStoreColors- Store RGB values into colormap cells.
XStoreColor- Store an RGB value into a single colormap cell.
XStoreNamedColor- Set a pixel color to the named color.
XFreeColors- Free colormap cells.

Creating and Freeing Pixmap

XCreatePixmap- Create a pixmap of a specified size.
XFreePixmap- Free all storage associated with specified pixmap.

Manipulating Graphics Contexts (GCs

XCreateGC- Create a new GC.
XCopyGC- Copy components from a source GC to a destination GC.
XChangeGC- Change the components in the specified GC.
XFreeGC- Free the specified GC.
XSetState- Set foreground, background, plane mask and function in GC.
XSetFunction- Set display function in specified GC.
XSetPlaneMask- Set the plane mask of the specified GC.
XSetForeground- Set the foregound of the specified GC.
XSetBackground- Set the backgound of the specified GC.
XSetLineAttributes- Set the line drawing components of the GC.
XSetDashes- Set the dashed line style components of specified GC.
XSetFillStyle- Set the fill style of the specified GC.
XSetFillRule- Set the fill rule of the specified GC.

¦ Copyright IBM Corp. 1985, 1991

XQueryBestSize- Get best size of tile, stipple, or cursor.
XQueryBestTile- Get best fill tile shape.
XQueryBestStipple- Get best stipple shape.
XSetTile- Set the fill tile of the specified GC.
XSetStipple- Set the stipple of the specified GC.
XSetTSOrigin- Set the tile or stipple origin of specified GC.
XSetFont- Set the current font of the specified GC.
XSetClipOrigin- Set the clip origin of the specified GC.
XSetClipMask- Set the clip_mask of specified GC to specified pixmap.
XSetClipRectangles- Set clip_mask of GC to list of rectangles.
XSetArcMode- Set the arc mode of the specified GC.
XSetSubwindowMode- Set subwindow mode of the specified GC.
XSetGraphicsExposures- Set graphics-exposure flag of specified GC.
XGContextFromGC- Obtain the GContext resource ID for GC.

Clearing and Copying Area

XClearArea- Clear a rectangular area of window.
XClearWindow- Clear the entire window.
XCopyArea- Copy drawable area between drawables of the same root and the same depth.
XCopyPlane- Copy single bit-plane of drawable.

Drawing Line

XDrawPoint- Draw a single point in specified drawable.
XDrawPoints- Draw multiple points in specified drawable.
XDrawLine- Draw a single line between two points in drawable.
XDrawLines- Draw multiple lines in the specified drawable.
XDrawSegments- Draw multiple line segments in specified drawable.
XDrawRectangle- Draw outline of single rectangle in drawable.
XDrawRectangles- Draw outline of multiple rectangles in drawable.
XDrawArc- Draw single arc in drawable.
XDrawArcs- Draw multiple arcs in specified drawable.

Filling Area

XFillRectangle- Fill single rectangular area in drawable.
XFillRectangles- Fill multiple rectangular areas in drawable.
XFillPolygon- Fill a polygon area in drawable.
XFillArc- Fill single arc in drawable.
XFillArcs- Fill multiple arcs in drawable.

Loading and Freeing Font

XLoadFont- Load a font.
XQueryFont- Get information about a loaded font.
XListFontsWithInfo- Get names and information about loaded fonts.
XFreeFontInfo- Free the font information array.
XLoadQueryFont- Loads and queries font in one operation.
XFreeFont- Unload font and free storage used by font.
XGetFontProperty- Get the specified font property.
XUnloadFont- Unload the specified font.
XListFonts- Get a list of available font names.
XFreeFontNames- Free a font name array.
XSetFontPath- Set the font search path.
XGetFontPath- Get the current font search path.
XFreeFontPath- Free data returned by **XGetFontPath**.

¦ Copyright IBM Corp. 1985, 1991

Querying Character String Size

XTextWidth- Get the width of an 8-bit character string.
XTextWidth16- Get the width of a 2-byte character string.
XTextExtents- Get bounding box of 1-byte character string.
XTextExtents16- Get bounding box of 2-byte character string.
XQueryTextExtents- Get 1-byte character string bounding box from server.
XQueryTextExtents16- Get 2-byte character string bounding box from server.

Drawing Tex

XDrawText- Draw 8-bit complex text in specified drawable.
XDrawText16- Draw 2-byte complex text in specified drawable.
XDrawString- Draw 8-bit text in specified drawable.
XDrawString16- Draw 2-byte text in specified drawable.
XDrawImageString- Draw 8-bit image text in specified drawable.
XDrawImageString16- Draw 2-byte image text in specified drawable.

Transferring Image

XPutImage- Put image from memory into rectangle in drawable.
XGetImage- Get image from rectangle in drawable.
XGetSubImage- Copy rectangle on display to image.

Manipulating Cursor

XCreateFontCursor- Create a cursor from a standard font.
XCreateGlyphCursor- Create a cursor from font glyphs.
XRecolorCursor- Change the color of a cursor.
XFreeCursor- Free a cursor.
XQueryBestCursor- Get useful cursor sizes.
XDefineCursor- Define a cursor for a window.
XUndefineCursor- Undefine a cursor for a window.

Handling Window Manager Function

XReparentWindow- Change the parent of a window.
XChangeSaveSet- Add or remove a window from the client's save-set.
XAddToSaveSet- Add a window to the client's save-set.
XRemoveFromSaveSet- Remove a window from the client's save-set.
XInstallColormap- Install a colormap.
XUninstallColormap- Uninstall a colormap.
XListInstalledColormaps- Get a list of currently installed colormaps.
XGrabPointer- Grab the pointer.
XGrabButton- Grab a mouse button.
XUngrabButton- Ungrab a mouse button.
XUngrabPointer- Ungrab the pointer.
XChangeActivePointerGrab- Change the active pointer grab.
XGrabKeyboard- Grab the keyboard.
XUngrabKeyboard- Ungrab the keyboard.
XGrabKey- Grab a single key of the keyboard.
XUngrabKey- Ungrab a key.
XAllowEvents- Allow events to be processed after a device is frozen.
XGrabServer- Grab the server.
XUngrabServer- Ungrab the server.
XWarpPointer- Move the pointer to arbitrary point on the screen.

¦ Copyright IBM Corp. 1985, 1991

XSetInputFocus- Set the input focus.
XGetInputFocus- Get the current input focus.
XChangePointerControl- Change the interactive feel of pointer device.
XGetPointerControl- Get the current pointer parameters.
XSetCloseDownMode- Change the close down mode of a client.
XKillClient- Remove a client.

Manipulating Keyboard Setting

XChangeKeyboardControl- Change keyboard settings.
XGetKeyboardControl- Get the current keyboard settings.
XAutoRepeatOn- Turn on keyboard auto-repeat.
XAutoRepeatOff- Turn off keyboard auto-repeat.
XBell- Set the volume of the bell.
XSetPointerMapping- Set the mapping of buttons on the pointer.
XGetPointerMapping- Get the mapping of buttons on the pointer.
XQueryKeymap- Get the state of the keyboard keys.
XDisplayKeycodes- Returns minimum and maximum number of keycodes supported by the specified display.
XGetKeyboardMapping- Get the mapping of symbols to keycodes.
XChangeKeyboardMapping- Change the mapping of symbols to keycodes.
XNewModifiermap- Create the **XModifierKeymap** structure.
XInsertModifiermapEntry- Add an entry to **XModifierKeymap** structure.
XDeleteModifiermapEntry- Delete an entry from **XModifierKeymap** structure.
XFreeModifiermap- Free **XModifierKeymap** structure.
XSetModifierMapping- Set keycodes to be modifiers.
XGetModiferMapping- Get keycodes to be modifiers.

Controlling the Screen Save

XSetScreenSaver- Set the screen saver.
XForceScreenSaver- Turn the screen saver on or off.
XActivateScreenSaver- Activate the screen saver.
XGetScreenSaver- Get the current screen saver settings.

Manipulating Hosts and Access Contro

XAddHost- Add a host.
XAddHosts- Add multiple hosts.
XListHosts- Get the list of hosts.
XRemoveHost- Remove a host.
XRemoveHosts- Remove multiple hosts.
XSetAccessControl- Change access control.
XEnableAccessControl- Enable access control.
XDisableAccessControl- Disable access control.

Handling Event

XSelectInput- Select events to be reported to the client.
XFlush- Flush the output buffer.
XSync- Flush the output buffer and wait until all requests are completed.
XEventQueued- Check the number of events in the event queue.
XPending- Return the number of events that are pending.
XNextEvent- Get the next event and remove it from the queue.
XPeekEvent- Peek at the event queue.

¦ Copyright IBM Corp. 1985, 1991

XIfEvent- Check event queue for specified event and remove it.
XCheckIfEvent- Check event queue for specified event without blocking.
XPeekIfEvent- Check event queue for specified event.
XWindowEvent- Remove next event that matches the specified window and mask.
XCheckWindowEvent- Remove next event that matches the specified window and mask without blocking.
XMaskEvent- Remove the next event that matches a specified mask.
XCheckMaskEvent- Remove the next event that matches a specified mask without blocking.
XCheckTypedEvent- Get the next event that matches event type.
XCheckTypedWindowEvent- Get the next event for specified window.
XPutBackEvent- Push event back to top of event queue.
XSendEvent- Send an event to a specified window.
XDisplayMotionBufferSize- Return the size of the motion buffer.
XGetMotionEvents- Get the motion history for specified window.

Enabling or Disabling Synchroniztio

(XSynchronize)()- Enable or disable synchronization.
(XSetAfterFunction)()- Set the previous after function.

Using Default Error Handlin

XSetErrorHandler- Set error handler.
XGetErrorText- Get error text for specified error code.
XGetErrorDatabaseText- Get error text from the error database.
XDisplayName- Get name of display currently being used.
XSetIOErrorHandler- Set error handler for fatal I/O errors.

Communicating with Window Manager

XSetStandardProperties- Specify a minumum set of properties.
XStoreName- Assign a name to a window.
XFetchName- Get the name of a window.
XSetIconName- Assign a name to an icon window.
XGetIconName- Get the name of an icon window.
XSetCommand- Set the value of the command atom.
XSetWMHints- Set the value of the window manager's hints atom.
XGetWMHints- Get the value of the window manager's hints atom.
XSetNormalHints- Set size hints for window in normal state.
XGetNormalHints- Get size hints for window in normal state.
XSetZoomHints- Set values of the zoom hints atom.
XGetZoomHints- Get values of the zoom hints atom.
XSetSizeHints- Set the values of type **WM_SIZE_HINTS** properties.
XGetSizeHints- Get the values of type **WM_SIZE_HINTS** properties.
XSetIconSizes- Set the values of icon size atom.
XGetIconSizes- Get the values of icon size atom.
XSetClassHint- Set the class of a window.
XGetClassHint- Get the class of a window.
XSetTransientForHint- Set **WM_TRANSIENT_FOR** property for window.
XGetTransientForHint- Get **WM_TRANSIENT_FOR** property for window.
XGetStandardColormap- Get colormap associated with specified atom.
XSetStandardColormap- Set colormap associated with specified atom.

Keyboard Event Function

XLookupKeysym- Translate keyboard event into keysym value.

¦ Copyright IBM Corp. 1985, 1991

XRefreshKeyboardMapping- Refresh stored modifier and keymap information.
XLookupString- Translate keyboard event into character string.
XRebindKeysym- Map character string to specified keysym and modifiers.
XStringToKeysym- Convert keysym name to keysym value.
XKeysymToString- Convert keysym value to keysym name.
XKeycodeToKeysym- Convert keycode to a keysym value.
XKeysymToKeycode- Convert keysym value to keycode.
XRebindCode- Change the keyboard mapping in keymap file.
XLookupMapping- Get mapping of keyboard event from keymap file.
XUseKeymap- Change keymap files.

Handling Default Geometry Colo

XGetDefault- Get default window options.
XParseGeometry- Parse standard window geometry options.
XResourceManagerString- Return the RESOURCE_MANAGER property from the screen of the root window of screen zero.
XGeometry- Parse window geometry given padding and font values.
XParseColor- Obtain RGB values from color name.

Manipulating Region

XPolygonRegion- Generate a region from points.
XClipBox- Generate the smallest enclosing rectangle in region.
XCreateRegion- Create a new empty region.
XSetRegion- Set the GC to the specified region.
XDestroyRegion- Free storage associated with specified region.
XOffsetRegion- Move specified region by specified amount.
XShrinkRegion- Reduce specified region by specified amount.
XIntersectRegion- Compute intersection of two regions.
XUnionRegion- Compute union of two regions.
XUnionRectWithRegion- Create a union of source region and rectangle.
XSubtractRegion- Subtract two regions.
XXorRegion- Get difference between union and intersection of regions.
XEmptyRegion- Determine if specified region is empty.
XEqualRegion- Determine if two regions are the same.
XPointInRegion- Determine if point lies in specified region.
XRectInRegion- Determine if rectangle lies in specified region.

Using Cut and Paste Buffer

XStoreBytes- Store data in first cut buffer.
XStoreBuffer- Store data in specified cut buffer.
XFetchBytes- Get data from first cut buffer.
XFetchBuffer- Get data from specified cut buffer.
XRotateBuffers- Rotate the cut buffers.

Querying Visual Type

XGetVisualInfo- Get list of visual information structures.
XMatchVisualInfo- Get visual information matching screen depth and class.
XVisualIDFromVisual- Return the visual ID for the specified visual type.

¦ Copyright IBM Corp. 1985, 1991

Manipulating Image

XCreateImage- Allocate memory for **XImage** structure.
XGetPixel- Get a pixel value in an image.
XPutPixel- Set a pixel value in an image.
XSubImage- Create image that is subsection of specified image.
XAddPixel- Increment each pixel in pixmap by constant value.
XDestroyImage- Free memory for **XImage** structure.

Manipulating Bitmap

XReadBitmapFile- Read in a bitmap from a file.
XWriteBitmapFile- Write out a bitmap to a file.
XCreatePixmapFromBitmapData- Create pixmap using bitmap data.
XCreateBitmapFromData- Include bitmap in **C** program.

Using the Resource Manage

XrmInitialize- Initialize the resource manager.
XrmUniqueQuark- Allocate a new quark.
Xpermalloc- Allocate memory which is never freed.
XrmStringToQuark- Convert character string to a quark.
XrmQuarkToString- Convert a quark to a character string.
XrmStringToQuarkList- Convert character strings to quark list.
XrmStringToBindingQuarkList- Convert strings to bindings and quarks.
XrmPutResource- Store resource into database.
XrmQPutResource- Store binding and quarks into database.
XrmPutStringResource- Store string resource into database.
XrmQPutStringResource- Store string binding and quarks into database.
XrmPutLineResource- Store single resource entry into database.
XrmGetResource- Retrieve a resource from a database.
XrmQGetResource- Retrieve a quark from a database.
XrmQGetSearchList- Get a resource search list of database levels.
XrmQGetSearchResource- Get a quark search list of database levels.
XrmMergeDatabases- Merge two databases.
XrmGetFileDatabase- Create a database from specified file.
XrmPutFileDatabase- Copy database into specified file.
XrmGetStringDatabase- Create a database from specified string.
XrmParseCommand- Store command options into a database.

Using the Context Manage

XSaveContext- Store data associated with window and context type.
XFindContext- Get data associated with window and context type.
XDeleteContext- Delete data associated with window and context type.
XUniqueContext- Allocate a new context.

¦ Copyright IBM Corp. 1985, 1991

2.4 Xlib Functions

Xlib functions are described in this chapter. The functions are arranged in alphabetical order.

Many functions return an integer resource ID. These integer resource IDs can be of type **Window, Font, Pixmap, Bitmap,** or **Cursor** (as defined in **/usr/include/X11/X.h**). Some functions return **Status**, which is an integer error code. If a function fails, it returns 0.

If a client does not want a request to execute asynchronously, it should be followed immediately by a call to **XSync**, which blocks until all previously-buffered asynchronous events have been sent and acted upon.

Subtopics

¦ Copyright IBM Corp. 1985, 1991

¦ Copyright IBM Corp. 1985, 1991

¦ Copyright IBM Corp. 1985, 1991

¦ Copyright IBM Corp. 1985, 1991

¦ Copyright IBM Corp. 1985, 1991

¦ Copyright IBM Corp. 1985, 1991

¦ Copyright IBM Corp. 1985, 1991

2.4.1 AllPlanes() or XAllPlanes()

AllPlanes()

unsigned long XAllPlanes()

Both the macro and the function return a value with all bits set on suitable for use in a plane argument to a procedure.

¦ Copyright IBM Corp. 1985, 1991

2.4.2 BitmapBitOrder or XBitmapBitOrder

BitmapBitOrder(*display*)

int XBitmapBitOrder(*display*)
Display **display*;

display Specifies the connection to the X Server.

Within each bitmap unit, the leftmost bit in the bitmap displayed on the screen is either the least-significant bit or the most-significant bit in the unit. Both the macro and the function return **LSBFirst** or **MSBFirst**.

¦ Copyright IBM Corp. 1985, 1991

2.4.3 BitmapPad or XBitmapPad

BitmapPad(*display*)

int XBitmapPad(*display*)
Display **display*;

display Specifies the connection to the X Server.

Each scanline must be padded to a multiple of bits returned by this macro or function.

¦ Copyright IBM Corp. 1985, 1991

2.4.4 BitmapUnit or XBitmapUnit

BitmapUnit(*display*)

int XBitmapUnit(*display*)
Display **display*;

display Specifies the connection to the X Server.

Both the macro and the function return the size of a bitmap's unit in bits. The scanline is calculated in multiples of this value.

¦ Copyright IBM Corp. 1985, 1991

2.4.5 BlackPixel or XBlackPixel

BlackPixel(*display, screen_number*)

unsigned long XBlackPixel(*display, screen_number*)
Display **display*;
int *screen_number*;

display Specifies the connection to the X Server.

screen_number Specifies the screen number of the display.

Both the macro and the function return the black pixel value for the screen specified.

¦ Copyright IBM Corp. 1985, 1991

2.4.6 BlackPixelOfScreen or XBlackPixelOfScreen

BlackPixelOfScreen(*screen*)

unsigned long XBlackPixelOfScreen(*screen*)
Screen **screen*;

screen Specifies the screen of the display.

Both the macro and the function return the black pixel value of the screen specified.

¦ Copyright IBM Corp. 1985, 1991

2.4.7 CellsOfScreen or XCellsOfScreen

CellsOfScreen(*screen*)

int XCellsOfScreen(*screen*)
Screen **screen*;

screen Specifies the screen of the display.

Both the macro and the function return the number of colormap cells of the screen specified.

¦ Copyright IBM Corp. 1985, 1991

2.4.8 ConnectionNumber or XConnectionNumber

ConnectionNumber(*display*)

int XConnectionNumber(*display*)
Display **display*;

display Specifies the connection to the X Server.

Both the macro and the function return a connection number, which is the file descriptor of the connection, for the display specified.

¦ Copyright IBM Corp. 1985, 1991

2.4.9 DefaultColormap or XDefaultColormap

DefaultColormap(*display, screen_number*)

Colormap XDefaultColormap(*display, screen_number*)
Display **display*;
int *screen_number*;

display Specifies the connection to the X Server.

screen_number Specifies the screen number of the display.

Both the macro and the function return the default colormap ID for allocation on the specified screen. This colormap should be used for most routine color allocations.

¦ Copyright IBM Corp. 1985, 1991

2.4.10 DefaultColormapOfScreen or XDefaultColormapOfScreen

DefaultColormapOfScreen(*screen*)

Colormap XDefaultColormapOfScreen(*screen*)
Screen **screen*;

screen Specifies the screen of the display.

Both the macro and the function return the default colormap of the screen specified.

¦ Copyright IBM Corp. 1985, 1991

2.4.11 DefaultDepth or XDefaultDepth

DefaultDepth(*display, screen_number*)

int XDefaultDepth(*display, screen_number*)
Display **display*;
int *screen_number*;

display Specifies the connection to the X Server.

screen_number Specifies the screen number of the display.

Both the macro and the function return the depth (number of planes) of the default root window for the screen specified. Other depths may also be supported on this screen. For more information on the depths available, see "Defining Visual Types" in topic 1.5.1.

¦ Copyright IBM Corp. 1985, 1991

2.4.12 DefaultDepthOfScreen or XDefaultDepthOfScreen

DefaultDepthOfScreen(*screen*)

int XDefaultDepthOfScreen(*screen*)
Screen **screen*;

screen Specifies the screen of the display.

Both the macro and the function return the default depth (number of planes) of the screen specified.

¦ Copyright IBM Corp. 1985, 1991

2.4.13 DefaultGC or XDefaultGC

DefaultGC(*display*, *screen_number*)

GC XDefaultGC(*display*, *screen_number*)
Display **display*;
int *screen_number*;

display Specifies the connection to the X Server.

screen_number Specifies the screen number of the display.

Both the macro and the function return the default graphics context (GC) of the default root window for the screen specified. This GC is created for the convenience of simple applications. It contains the default GC components with the foreground and background pixel values initialized to the black and white pixels, respectively, for the screen. This GC can be modified. See "Manipulating Graphics Context or State" in topic 1.7.8.

¦ Copyright IBM Corp. 1985, 1991

2.4.14 DefaultGCOfScreen or XDefaultGCOfScreen

DefaultGCOfScreen(*screen*)

GC XDefaultGCOfScreen(*screen*)
Screen **screen*;

screen Specifies the screen of the display.

Both the macro and the function return the default graphics context (GC) of the screen specified.

¦ Copyright IBM Corp. 1985, 1991

2.4.15 DefaultRootWindow or XDefaultRootWindow

DefaultRootWindow(*display*)

Window XDefaultRootWindow(*display*)
Display **display*;

display Specifies the connection to the X Server.

Both the macro and the function return the root window for the default screen.

¦ Copyright IBM Corp. 1985, 1991

2.4.16 DefaultScreen or XDefaultScreen

DefaultScreen(*display*)

int XDefaultScreen(*display*)
Display **display*;

display Specifies the connection to the X Server.

Both the macro and the function return the default screen referenced in the **XOpenDisplay** routine. Use this macro or function to retrieve the screen number in applications that use a single screen only. (1)

(1) AIX X-Windows supports only one screen.

¦ Copyright IBM Corp. 1985, 1991

2.4.17 DefaultScreenOfDisplay or XDefaultScreenOfDisplay

DefaultScreenOfDisplay(*display*)

Screen *XDefaultScreenOfDisplay(*display*)
Display **display*;

display Specifies the connection to the X Server.

Both the macro and the function return the default screen of the display specified.

¦ Copyright IBM Corp. 1985, 1991

2.4.18 DefaultVisual or XDefaultVisual

DefaultVisual(*display, screen_number*)

Visual *XDefaultVisual(*display, screen_number*)
Display **display*;
int *screen_number*;

display Specifies the connection to the X Server.

screen_number Specifies the number screen of the display.

Both the macro and the function return the default visual type for the screen specified. See "Determining the Appropriate Visual" in topic 1.5.2.

¦ Copyright IBM Corp. 1985, 1991

2.4.19 DefaultVisualOfScreen or XDefaultVisualOfScreen

DefaultVisualOfScreen(*screen*)

Visual *XDefaultVisualOfScreen(*screen*)
Screen **screen*;

screen Specifies the screen of the display.

Both the macro and the function return the default visual of the screen specified. See "Determining the Appropriate Visual" in topic 1.5.2.

¦ Copyright IBM Corp. 1985, 1991

2.4.20 DisplayCells or XDisplayCells

DisplayCells(*display, screen_number*)

int XDisplayCells(*display, screen_number*)
Display **display*;
int *screen_number*;

display Specifies the connection to the X Server.

screen_number Specifies the screen number of the display.

Both the macro and the function return the number of entries in the default colormap.

¦ Copyright IBM Corp. 1985, 1991

2.4.21 DisplayHeight or XDisplayHeight

DisplayHeight(*display, screen_number*)

int XDisplayHeight(*display, screen_number*)
Display **display*;
int *screen_number*;

display Specifies the connection to the X Server.

screen_number Specifies the number screen of the display.

Both the macro and the function return an integer that describes the height of the screen in pixels.

¦ Copyright IBM Corp. 1985, 1991

2.4.22 DisplayHeightMM or XDisplayHeightMM

DisplayHeightMM(*display, screen_number*)

int XDisplayHeightMM(*display, screen_number*)
Display **display*;
int *screen_number*;

display Specifies the connection to the X Server.

screen_number Specifies the screen number of the display.

Both the macro and the function return an integer that describes the height of the screen in millimeters.

¦ Copyright IBM Corp. 1985, 1991

2.4.23 DisplayOfScreen or XDisplayOfScreen

DisplayOfScreen(*screen*)

Display *XDisplayOfScreen(*screen*)
Screen **screen*;

screen Specifies the screen of the display.

Both the macro and the function return the display of the screen specified.

¦ Copyright IBM Corp. 1985, 1991

2.4.24 DisplayPlanes or XDisplayPlanes

DisplayPlanes(*display, screen_number*)

int XDisplayPlanes(*display, screen_number*)
Display **display*;
int *screen_number*;

display Specifies the connection to the X Server.

screen_number Specifies the screen number of the display.

Both the macro and the function return the depth (number of planes) of the root window of the screen specified. For more information, see "Determining the Appropriate Visual" in topic 1.5.2.

¦ Copyright IBM Corp. 1985, 1991

2.4.25 DisplayString or XDisplayString

DisplayString(*display*)

char *XDisplayString(*display*)
Display **display*;

display Specifies the connection to the X Server.

Both the macro and the function obtain the string passed to **XOpenDisplay** when the current display was opened. If the string was **NULL**, the value of the **DISPLAY** environment when the current display was opened is returned. This is useful with applications that invoke the **fork** system call and want to open a new connection to the same display from the child process.

¦ Copyright IBM Corp. 1985, 1991

2.4.26 DisplayWidth or XDisplayWidth

DisplayWidth(*display, screen_number*)

int XDisplayWidth(*display, screen_number*)
Display **display*;
int *screen_number*;

display Specifies the connection to the X Server.

screen_number Specifies the screen number of the display.

Both the macro and the function return an integer that describes the width of the screen in pixels.

¦ Copyright IBM Corp. 1985, 1991

2.4.27 DisplayWidthMM or XDisplayWidthMM

DisplayWidthMM(*display, screen_number*)

int XDisplayWidthMM(*display, screen_number*)
Display **display*;
int *screen_number*;

display Specifies the connection to the X Server.

screen_number Specifies the screen number of the display.

Both the macro and the function return an integer that describes the width of the screen in millimeters.

¦ Copyright IBM Corp. 1985, 1991

2.4.28 DoesBackingStore or XDoesBackingStore

DoesBackingStore(*screen*)

int XDoesBackingStore(*screen*)
Screen **screen*;

screen Specifies the screen of the display.

Both the macro and the function return a value indicating if the screen supports backing stores.

The value returned can be **WhenMapped**, **NotUseful**, or **Always**. For information about these values, see page 1.5.3.

¦ Copyright IBM Corp. 1985, 1991

2.4.29 DoesSaveUnders or XDoesSaveUnders

DoesSaveUnders(*screen*)

Bool XDoesSaveUnders(*screen*)
Screen **screen*;

screen Specifies the screen of the display.

Both the macro and the function return a Boolean value indicating if the specified screen supports save unders.

If **True** is returned, the screen supports save unders.
If **False** is returned, the screen does not support save unders.

For information about save unders, see page 1.5.3.

¦ Copyright IBM Corp. 1985, 1991

2.4.30 EventMaskOfScreen or XEventMaskOfScreen

EventMaskOfScreen(*screen*)

long XEventMaskOfScreen(*screen*)
Screen **screen*;

screen Specifies the screen of the display.

Both the macro and the function return the initial root event mask for the screen.

¦ Copyright IBM Corp. 1985, 1991

2.4.31 HeightMMOfScreen or XHeightMMOfScreen

HeightMMOfScreen(*screen*)

int XHeightMMOfScreen(*screen*)
Screen **screen*;

screen Specifies the screen of the display.

Both the macro and the function return an integer that describes the height of the screen specified in millimeters.

¦ Copyright IBM Corp. 1985, 1991

2.4.32 HeightOfScreen or XHeightOfScreen

HeightOfScreen(*screen*)

int XHeightOfScreen(*screen*)
Screen **screen*;

screen Specifies the screen of the display.

Both the macro and the function return an integer that describes the height of the screen specified in pixels.

¦ Copyright IBM Corp. 1985, 1991

2.4.33 ImageByteOrder or XImageByteOrder

ImageByteOrder(*display*)

int XImageByteOrder(*display*)
Display **display*;

display Specifies the connection to the X Server.

Both the macro and the function specify the required byte order for images for each scanline unit in **XYFormat** (bitmap) or for each pixel value in **ZFormat**. This macro and function can return **LSBFirst** or **MSBFirst**.

¦ Copyright IBM Corp. 1985, 1991

2.4.34 IsCursorKey

IsCursorKey(*keysym*)

IsCursorKey returns **True** if the keysym is on the cursor key.

¦ Copyright IBM Corp. 1985, 1991

2.4.35 IsFunctionKey

IsFunctionKey(*keysym*)

IsFunctionKey returns **True** if the keysym is on the function keys.

¦ Copyright IBM Corp. 1985, 1991

2.4.36 IsKeypadKey

IsKeypadKey(*keysym*)

IsKeypadKey returns **True** if the keysym is on the keypad.

¦ Copyright IBM Corp. 1985, 1991

2.4.37 IsMiscFunctionKey

IsMiscFunctionKey (*keysym*)

IsMiscFunctionKey returns **True** if the keysym is on the miscellaneous function keys.

¦ Copyright IBM Corp. 1985, 1991

2.4.38 IsModifierKey

IsModifierKey(*keysym*)

IsModifierKey returns **True** if the keysym is on the modifier keys.

¦ Copyright IBM Corp. 1985, 1991

2.4.39 IsPFKey

IsPFKey(*keysym*)

IsPFKey returns **True** if the keysym is on the **PF** keys.

¦ Copyright IBM Corp. 1985, 1991

2.4.40 LastKnownRequestProcessed or XLastKnownRequestProcessed

LastKnownRequestProcessed(*display*)

int XLastKnownRequestProcessed(*display*)
Display **display*;

display Specifies the connection to the X Server.

Both the macro and the function extract the full serial number of the last request known by X-Windows to have been processed by the X Server. This number is automatically set when replies, events, and errors are received.

¦ Copyright IBM Corp. 1985, 1991

2.4.41 MaxCmapsOfScreen or XMaxCmapsOfScreen

MaxCmapsOfScreen(*screen*)

int XMaxCmapsOfScreen(*screen*)
Screen **screen*;

screen Specifies the screen of the display.

Both the macro and the function return the maximum number of colormaps supported by the screen specified.

¦ Copyright IBM Corp. 1985, 1991

2.4.42 MinCmapsOfScreen or XMinCmapsOfScreen

MinCmapsOfScreen(*screen*)

int XMinCmapsOfScreen(*screen*)
Screen **screen*;

screen Specifies the screen of the display.

Both the macro and the function return the minimum number of colormaps supported by the screen specified.

¦ Copyright IBM Corp. 1985, 1991

2.4.43 NextRequest or XNextRequest

NextRequest(*display*)

int XNextRequest(*display*)
Display **display*;

display Specifies the connection to the X Server.

Both the macro and the function extract the full serial number to be used for the next request. Serial numbers are maintained separately for each display connection.

¦ Copyright IBM Corp. 1985, 1991

2.4.44 PlanesOfScreen or XPlanesOfScreen

PlanesOfScreen(*screen*)

int XPlanesOfScreen(*screen*)
Screen **screen*;

screen Specifies the screen of the display.

Both the macro and the function return the number of planes (depth) in the screen specified.

¦ Copyright IBM Corp. 1985, 1991

2.4.45 ProtocolRevision or XProtocolRevision

ProtocolRevision(*display*)

int XProtocolRevision(*display*)
Display **display*;

display Specifies the connection to the X Server.

Both the macro and the function return the minor protocol revision number (zero) of the X Server associated with the display.

¦ Copyright IBM Corp. 1985, 1991

2.4.46 ProtocolVersion or XProtocolVersion

ProtocolVersion(*display*)

int XProtocolVersion(*display*)
Display **display*;

display Specifies the connection to the X Server.

Both the macro and the function return the major version number (11) of the X-Windows protocol associated with the display.

¦ Copyright IBM Corp. 1985, 1991

2.4.47 QLength or XQLength

QLength(*display*)

int XQLength(*display*)
Display **display*;

display Specifies the connection to the X Server.

Both the macro and the function return the length of the event queue for the display. There may be other events that have not been read into the queue yet.

¦ Copyright IBM Corp. 1985, 1991

2.4.48 RootWindow or XRootWindow

RootWindow(*display, screen_number*)

Window XRootWindow(*display, screen_number*)
Display **display*;
int *screen_number*;

display Specifies the connection to the X Server.

screen_number Specifies the screen number of the display.

Both the macro and the function return the root window. This is useful with functions that take a parent window as an argument.

¦ Copyright IBM Corp. 1985, 1991

2.4.49 RootWindowOfScreen or XRootWindowOfScreen

RootWindowOfScreen(*screen*)

Window XRootWindowOfScreen(*screen*)
Screen **screen*;

screen Specifies the screen of the display.

Both the macro and the function return the root window of the screen specified.

¦ Copyright IBM Corp. 1985, 1991

2.4.50 ScreenCount or XScreenCount

ScreenCount(*display*)

int XScreenCount(*display*)
Display **display*;

display Specifies the connection to the X Server.

Both the macro and the function return the number of available screens.

¦ Copyright IBM Corp. 1985, 1991

2.4.51 ScreenOfDisplay or XScreenOfDisplay

ScreenOfDisplay(*display, screen_number*)

Screen *XScreenOfDisplay(*display, screen_number*)
Display **display*;
int *screen_number*;

display Specifies the connection to the X Server.

screen_number Specifies the screen number of the display.

Both the macro and the function return a pointer to the screen of the display specified.

¦ Copyright IBM Corp. 1985, 1991

2.4.52 ServerVendor or XServerVendor

ServerVendor(*display*)

char *XServerVendor(*display*)
Display **display*;

display Specifies the connection to the X Server.

Both the macro and the function return a pointer to a null-terminated string which provides some identification of the owner of the X Server implementation.

¦ Copyright IBM Corp. 1985, 1991

2.4.53 VendorRelease or XVendorRelease

VendorRelease(*display*)

int XVendorRelease(*display*)
Display **display*;

display Specifies the connection to the X Server.

Both the macro and the function return a number related to a vendor's release of the X Server.

¦ Copyright IBM Corp. 1985, 1991

2.4.54 WhitePixel or XWhitePixel

WhitePixel(*display, screen_number*)

unsigned long XWhitePixel(*display, screen_number*)
Display **display*;
int *screen_number*;

display Specifies the connection to the X Server.

screen_number Specifies the screen number of the display.

Both the macro and the function return the white pixel value for the screen specified.

¦ Copyright IBM Corp. 1985, 1991

2.4.55 WhitePixelOfScreen or XWhitePixelOfScreen

WhitePixelOfScreen(*screen*)

unsigned long XWhitePixelOfScreen(*screen*)
Screen **screen*;

screen Specifies the screen of the display.

Both the macro and the function return the white pixel value of the screen specified.

¦ Copyright IBM Corp. 1985, 1991

2.4.56 WidthMMOfScreen or XWidthMMOfScreen

WidthMMOfScreen(*screen*)

int XWidthMMOfScreen(*screen*)
Screen **screen*;

screen Specifies the screen of the display.

Both the macro and the function return an integer that describes the width of the screen specified in millimeters.

¦ Copyright IBM Corp. 1985, 1991

2.4.57 WidthOfScreen or XWidthOfScreen

WidthOfScreen(*screen*)

int XWidthOfScreen(*screen*)
Screen **screen*;

screen Specifies the screen of the display.

Both the macro and the function return an integer that describes the width of the screen specified in pixels.

¦ Copyright IBM Corp. 1985, 1991

2.4.58 XActivateScreenSaver

XActivateScreenSaver(*display*)
Display **display*;

display Specifies the connection to the X Server.

XActivateScreenSaver activates the screen saver. See **XSetScreenSaver** for more information.

¦ Copyright IBM Corp. 1985, 1991

2.4.59 XAddHost

XAddHost(*display, host*)
Display **display*;
XHostAddress **host*;

display Specifies the connection to the X Server.

host Specifies the network address of the host machine.

XAddHost adds the specified host to the access control list for that display. The display (server) and the program (client) must be on the same host. See "Controlling Host Access" in topic 1.10.6.

This routine can generate the event errors **BadAlloc** and **BadValue**.

¦ Copyright IBM Corp. 1985, 1991

2.4.60 XAddHosts

XAddHosts(*display, hosts, num_hosts*)
Display **display*;
XHostAddress **hosts*;
int *num_hosts*;

display Specifies the connection to the X Server.

hosts Specifies each host to be added.

num_hosts Specifies the number of hosts to be added.

XAddHosts adds each specified host to the access control list for that display. The display (server) and the program (client) must be on the same host. See "Controlling Host Access" in topic 1.10.6.

This routine can generate the event errors **BadAlloc** and **BadValue**.

¦ Copyright IBM Corp. 1985, 1991

2.4.61 XAddPixel

int XAddPixel(*ximage, value*)
XImage **ximage*;
int *value;*

ximage Specifies a pointer to the image.

value Specifies the value to be added.

XAddPixel adds a value to every pixel in an image. Use this function to manipulate the base pixel value for allocating color resources to an image. See "Transferring Images Between Client and Server" in topic 1.9.3 for information about the **XImage** data structure.

¦ Copyright IBM Corp. 1985, 1991

2.4.62 XAddToSaveSet

XAddToSaveSet(*display, window_add*)
Display **display*;
Window *window_add*;

display Specifies the connection to the X Server.

window_add Specifies the window ID of the window to be added.

XAddToSaveSet adds the window and the children of the window specified to the client save-set. The specified window must be created by another client. The X Server automatically removes the windows from the save-set when the specified window is destroyed.

This routine can generate the event errors **BadMatch** and **BadWindow**.

¦ Copyright IBM Corp. 1985, 1991

2.4.63 XAllocColor

Status XAllocColor(*display, cmap, screen_def_return*)
Display **display*;
Colormap *cmap*;
XColor **screen_def_return*;

display	Specifies the connection to the X Server.
cmap	Specifies the colormap ID.
screen_def_return	Returns the values in the colormap.

XAllocColor returns the pixel value indicating the closest available color supported by the hardware. It allocates a read-only colormap entry corresponding to the closest red, green, and blue values supported by the hardware. The colormap cells are shared among clients. When the last client deallocates a shared cell, the colormap cell is deallocated. For information about the **XColor** data structure, see "Creating Colormaps" in topic 1.7.1.

XAllocColor returns a zero if it is unsuccessful. Otherwise, **XAllocColor** returns a non-zero.

XAllocColor can generate the event errors **BadAlloc** and **BadColor**.

¦ Copyright IBM Corp. 1985, 1991

2.4.64 XAllocColorCells

Status XAllocColorCells(*display, cmap, contig, plane_masks_return, nplanes, pixels_return, ncolors*)
Display **display*;
Colormap *cmap*;
Bool *contig*;
unsigned long *plane_masks_return*[];
unsigned int *nplanes*;
unsigned long *pixels_return*[];
unsigned int *ncolors*;

display	Specifies the connection to the X Server.
cmap	Specifies the colormap ID.
contig	Specifies a Boolean value.
plane_masks_return	Returns an array of plane masks.
nplanes	Specifies the number of plane masks returned in the *plane_masks_return*.
pixels_return	Returns an array of pixel values.
ncolors	Specifies the number of pixel values returned in the *pixels_return* array.

XAllocColorCells allocates color cells. The number of colors must be positive and the number of planes must be non-negative.

If the planes must be contiguous, set *contig* to one. If the planes do not need to be contiguous, set *contig* to zero.

If *nplanes* and *ncolors* are requested, then *nplanes* plane masks and *ncolors* pixels are returned. No mask will have any bits in common with any other mask or with any of the pixels. By combining masks and pixels, **ncolors* 2 (nplanes)** distinct pixel values can be produced. These pixel values are allocated writable by the request.

If *contig* is **True** and all masks are combined,

A single contiguous set of bits will be formed for **GrayScale** or **PseudoColor**. Each mask will have one bit.
Three contiguous sets of bits (one within each pixel subfield) will b formed for **DirectColor**. Each mask will have 3 bits.

The RGB values of the allocated entries are undefined. See "Determining the Appropriate Visual" in topic 1.5.2.

XAllocColorCells can generate the event errors **BadAlloc**, **BadColor**, and **BadValue**.

¦ Copyright IBM Corp. 1985, 1991

2.4.65 XAllocColorPlanes

Status XAllocColorPlanes(*display, cmap, contig, pixels_return, ncolors, nreds,*
nblues, rmask_return, gmask_return, bmask_r
Display **display*;
Colormap *cmap*;
Bool *contig*;
unsigned long *pixels_return*[];
int *ncolors*;
int *nreds, ngreens, nblues*;
unsigned long **rmask_return, *gmask_return, *bmask_return*;

display Specifies the connection to the X Server.

cmap Specifies the colormap ID.

contig Specifies a Boolean value.

pixels_return Returns an array of pixel values.

ncolors Specifies the number of pixel values that are to be returned in the *pixels_return* array. This value must be a positive value.

nreds Specifies the number of red colors or shades. This value must be a positive value.

ngreens Specifies the number of green blue colors or shades. This value must be a positive value.

nblues Specifies the number of blue colors or shades. This value must be a positive value.

rmask_return Returns bit masks for the red planes.

gmask_return Returns bit masks for the green planes.

bmask_return Returns bit masks for the blue planes.

XAllocColorPlanes allocates color planes. It returns the pixel values in the *pixels_return* array.

If the planes must be contiguous, set *contig* to a value of one. If the planes do not need to be contiguous, set *contig* to the value of zero.

If *ncolors* colors, *nreds* reds, *ngreens* greens, and *nblues* blues are requested, *ncolors* pixels are returned. The masks returned have *nreds*, *ngreens*, and *nblues* bits set respectively.

Each mask will have a contiguous set of bits if contiguous is **True**. No mask will have any bits in common with any other mask or with any of the pixels.

For **DirectColor** and **PseudoColor**, each mask will lie within the corresponding pixel subfield. Distinct pixels values can be produced by combining subsets of masks with pixels.

ncolors* 2((nreds+ngreens+nblues))

These combinations are allocated by the request. However, the colormap

¦ Copyright IBM Corp. 1985, 1991

only contains the following:

ncolors* 2(nreds) independent red entries,

ncolors* 2(ngreens) independent green entries,

ncolors* 2(nblues) independent blue entries.

When the colormap entry for a pixel value is changed with **XStoreColors** or **XStoreNamedColor**, the pixel is decomposed according to the masks, and the corresponding independent entries are updated.

XAllocColorPlanes can generate the event errors **BadAlloc**, **BadColor**, and **BadValue**.

¦ Copyright IBM Corp. 1985, 1991

2.4.66 XAllocNamedColor

Status XAllocNamedColor(*display, cmap, color_name, screen_def_return, exact_def_return*)
Display **display*;
Colormap *cmap*;
char **color_name*;
XColor **screen_def_return, *exact_def_return*;

display	Specifies the connection to the X Server.
cmap	Specifies the colormap ID.
color_name	Specifies the color name string for the color definition structure to be returned. The color name is not case-sensitive.
screen_def_return	Returns the values used in the colormap.
exact_def_return	Returns the true pixel values for the closest available color provided by the hardware for the color name specified.

XAllocNamedColor obtains the color definition structure for a specified color. It determines the correct color or shade for the screen. For information about the **XColor** data structure, see "Creating Colormaps" in topic 1.7.1.

XAllocNamedColor returns a zero if it is unsuccessful. Otherwise, it returns a non-zero, the exact database definition, and the closest available color supported by the hardware.

XAllocNamedColor can generate the event errors **BadAlloc**, **BadColor**, and **BadName**.

¦ Copyright IBM Corp. 1985, 1991

2.4.67 XAllowEvents

XAllowEvents(*display, event_mode, time*)
Display **display*;
int *event_mode*;
Time *time*;

display Specifies the connection to the X Server.

event_mode Specifies the event mode.

time Specifies the time in a timestamp, which is expressed in milliseconds, or **CurrentTime**.

XAllowEvents releases some queued events if the client has caused a device to freeze. This function has no effect if the specified time is earlier than the last-grab time of the most recent active grab for the client, or if the specified time is later than the current X Server time. Refer to "Processing Events" in topic 1.11.4 for more information.

It is possible for both a pointer and a keyboard to be grabbed simultaneously by the same client or different clients. If a device is frozen on behalf of either grab, no event processing is performed for the device. It is also possible for a single device to be frozen because of both grabs. In this case, the device must be released on behalf of both grabs before events can be processed.

The following describes the processing that occurs according to the constant set in the *event_mode* argument.

AsyncPointer If the pointer is frozen by the client, the pointer event processing continues as usual. If the pointer is frozen by the client on behalf of two separate grabs, **AsyncPointer** releases both.

AsyncPointer has no effect if the pointer is not frozen by the client, but the pointer does need to be grabbed by the client.

SyncPointer If the pointer is frozen and actively grabbed by the client, pointer event processing continues normally until the next **ButtonPress** or **ButtonRelease** event is reported to the client. At this time, the pointer appears to be frozen again. However, if the reported event causes the pointer grab to be released, the pointer is not frozen.

SyncPointer has no effect if the pointer is not frozen by the client or if the pointer is not grabbed by the client.

ReplayPointer If the pointer is actively grabbed by the client and frozen as the result of an event having been sent to the client, either by **XGrabButton** or a previous **XAllowEvents** with mode (**SyncPointer**, but not from **XGrabPointer**), the pointer grab is released and that event is completely reprocessed. This time, however, the **XAllowEvents** ignores any passive grabs at or above (toward the root) the *grab_window* of the grab just

¦ Copyright IBM Corp. 1985, 1991

released.

ReplayPointer has no effect if the pointer is not grabbed by the client or if the pointer is not frozen as the result of an event.

AsyncKeyboard

If the keyboard is frozen by the client, the keyboard event processing continues as usual. If the keyboard is frozen twice by the client on behalf of two separate grabs, **AsyncKeyboard** releases for both.

AsyncKeyboard has no effect if the keyboard is not frozen by the client, and the keyboard does not need to be grabbed by the client.

SyncKeyboard

If the keyboard is frozen and actively grabbed by the client, keyboard event processing continues as usual until the next **KeyPress** or **KeyRelease** event is reported to the client. At this time, the keyboard again appears to be frozen. However, if the reported event causes the keyboard grab to be released, the keyboard does not freeze.

SyncKeyboard has no effect if the keyboard is not frozen by the client or if the keyboard is not grabbed by the client.

ReplayKeyboard

If the keyboard is actively grabbed by the client and is frozen as the result of an event having been sent to the client, either by **XGrabKey** or a previous **XAllowEvents** with mode (**SyncKeyboard**, but not from an **XGrabKeyboard**), the keyboard grab is released and that event is completely reprocessed. This time, however, **XAllowEvents** ignores any passive grabs at or above (toward the root) the *grab_window* of the grab just released.

ReplayKeyboard has no effect if the keyboard is not grabbed by the client or if the keyboard is not frozen as the result of an event.

SyncBoth

If both pointer and keyboard are frozen by the client, event processing (for both devices) continues normally until the next **ButtonPress**, **ButtonRelease**, **KeyPress**, or **KeyRelease** event is reported to the client for a grabbed device (button event for the pointer, key event for the keyboard), at which time the devices again appear to freeze. However, if the reported event causes the grab to be released, then the devices do not freeze. If the other device is still grabbed, then a subsequent event will still cause both devices to freeze.

SyncBoth has no effect unless both pointer and keyboard are frozen by the client. If the pointer or keyboard is frozen twice by the client on behalf of two separate grabs, **SyncBoth** releases for both grabs (but subsequent holds on **SyncBoth** will freeze each device only once).

¦ Copyright IBM Corp. 1985, 1991

AsyncBoth If the pointer and the keyboard are frozen by the client, event processing (for both devices) continues normally. If a device is frozen twice by the client on behalf of two separate grabs, **AsyncBoth** releases for both.

AsyncBoth has no effect unless both pointer and keyboard are frozen by the client.

AsyncPointer, **SyncPointer**, and **ReplayPointer** have no effect on the processing of keyboard events. **AsyncKeyboard**, **SyncKeyboard**, and **ReplayKeyboard** have no effect on the processing of pointer events.

XAllowEvents can generate the event error **BadValue**.

¦ Copyright IBM Corp. 1985, 1991

2.4.68 XAutoRepeatOff

XAutoRepeatOff(*display*)
Display **display*;

display Specifies the connection to the X Server.

XAutoRepeatOff turns off auto-repeat for the keyboard on the specified display.

¦ Copyright IBM Corp. 1985, 1991

2.4.69 XAutoRepeatOn

XAutoRepeatOn(*display*)
Display **display*;

display Specifies the connection to the X Server.

XAutoRepeatOn turns on auto-repeat for the keyboard on the specified display.

¦ Copyright IBM Corp. 1985, 1991

2.4.70 XBell

XBell(*display, percent*)
Display **display*;
int *percent*;

display Specifies the connection to the X Server.

percent Specifies the base volume for the bell. The volume can range from -100 to 100 inclusive.

XBell rings the bell on the keyboard on the specified display, if possible. The volume specified is relative to the base volume for the keyboard. If the value for the *percent* argument is not in the range -100 to 100 (inclusive) an error is generated. The volume at which the bell is rung when the *percent* argument is non-negative is:

```
base - [(base * percent) / 100] + percent
```

The volume at which the bell is rung when the *percent* argument is negative is:

```
base + [(base * percent) / 100]
```

To change the base volume of the bell for this keyboard, use **XChangeKeyboardControl**.

¦ Copyright IBM Corp. 1985, 1991

2.4.71 XChangeActivePointerGrab

XChangeActivePointerGrab(*display, event_mask, cursor, time*)
Display **display*;
unsigned int *event_mask*;
Cursor *cursor*;
Time *time*;

display Specifies the connection to the X Server.

event_mask Specifies the pointer events to be reported to the client. The event mask is a bitwise inclusive OR of valid pointer event mask bits.

cursor Specifies the cursor to be displayed or **None**.

time Specifies the time in a timestamp, which is expressed in milliseconds, or **CurrentTime**.

XChangeActivePointerGrab changes the specified dynamic parameters if the pointer is actively grabbed by the client and if the specified time is no earlier than the last-pointer-grab time and no later than the current X Server time. **XChangeActivePointerGrab** has no effect on the passive parameters of an **XGrabButton**.

The *event_mask* can be one of the following:

ButtonPressMask	**Button1MotionMask**	**PointerMotionHintMask**
ButtonReleaseMask	**Button2MotionMask**	**PointerMotionMask**
EnterWindowMask	**Button3MotionMask**	**ButtonMotionMask**
LeaveWindowMask	**Button4MotionMask**	**KeymapStateMask**
	Button5MotionMask	

See "Defining Event Masks" in topic 1.11.3 for more information.

XChangeActivePointerGrab can generate the event error **BadCursor**.

¦ Copyright IBM Corp. 1985, 1991

2.4.72 XChangeGC

XChangeGC(*display, gc, valuemask_change, values*)
Display **display*;
GC *gc;*
unsigned long *valuemask_change;*
XGCValues **values;*

display	Specifies the connection to the X Server.
gc	Specifies the graphics context.
valuemask_change	Specifies the components in the graphics context to be changed using information in the **XGCValues** structure. This argument is the bitwise inclusive OR of one or more of the valid **GC** component masks.
values	Specifies a pointer to the **XGCValues** structure.

XChangeGC changes the components specified by the the *valuemask_change* argument in the graphics context. The order in which components are verified and altered is server-dependent. If an error is generated, a subset of the components may have been altered. See "Manipulating Graphics Context or State" in topic 1.7.8 for information on graphics context components and the the **XGCValues** data structure.

Components are verified according to mask values defined in **<X11/X.h>**.

XChangeGC can generate the event errors **BadAlloc**, **BadFont**, **BadGC**, **BadMatch**, **BadPixmap**, and **BadValue**.

¦ Copyright IBM Corp. 1985, 1991

2.4.73 XChangeKeyboardControl

XChangeKeyboardControl(*display, value_mask, values*)
Display **display*;
unsigned long *value_mask*;
XKeyboardControl **values*;

display Specifies the connection to the X Server.

value_mask Specifies one value, from the least-significant bit to the most-significant bit, for each one bit in the mask. These values are associated with the set of keys for the keyboard specified previously.

values Specifies a pointer to the structure **XKeyboardControl**.

XChangeKeyboardControl controls the keyboard characteristics defined by the **XKeyboardControl** structure. See "Manipulating Keyboard Settings" in topic 1.10.4 for information about the **XKeyboardControl** data structure.

XChangeKeyboardControl can generate event errors **BadMatch** and **BadValue**.

¦ Copyright IBM Corp. 1985, 1991

2.4.74 XChangeKeyboardMapping

XChangeKeyboardMapping(*display, first_keycode, keysyms_per_keycode, keysyms, num_codes*)
Display **display*;
int *first_keycode*;
int *keysyms_per_keycode*;
KeySym **keysyms*;
int *num_codes*;

display	Specifies the connection to the X Server.
first_keycode	Specifies the first keycode to be changed.
keysyms_per_keycode	Specifies the keysyms to be used.
keysyms	Specifies a pointer to an array of keysyms.
num_codes	Specifies the number of keycodes that are to be changed.

XChangeKeyboardMapping defines the symbols for the specified number of keycodes starting with the first keycode indicated in the *first_keycode* variable. The symbols for keycodes outside this range remained unchanged.

The number of elements in the *keysyms* array must be a multiple of *keysyms_per_keycode*. Otherwise, an error is generated.

The specified *first_keycode* must be greater than or equal to *min_keycode* supplied at connection setup and stored in the **Display** structure. Otherwise, an error is generated.

In addition, the following expression must be less than or equal to *max_keycode* as returned in the connection setup

```
first_keycode + (num_codes / keysyms_per_keycode) - 1
```

The **KeySym** number N, counting from zero, for keycode K has an index, counting from zero, of the following in keysyms:

```
(K - first_keycode) * keysyms_per_keycode + N
```

The specified *keysyms_per_keycode* can be chosen arbitrarily by the client to be large enough to hold all desired symbols. The **KeySym** value of **NoSymbol** should be used for undefined elements in individual keycodes. **NoSymbol** can appear in nontrailing positions of the effective list for a keycode.

XChangeKeyboardMapping generates a **MappingNotify** event. The X Server does not need to interpret this mapping, but merely should store it for reading and writing by clients.

XChangeKeyboardMapping can generate the event errors **BadAlloc, BadLength**, and **BadValue**.

¦ Copyright IBM Corp. 1985, 1991

2.4.75 XChangePointerControl

XChangePointerControl(*display, do_accel, do_threshold, accel_numerator, accel_denominator, threshold*)
Display **display*;
Bool *do_accel, do_threshold*;
int *accel_numerator, accel_denominator*;
int *threshold*;

display	Specifies the connection to the X Server.
do_accel	Specifies a Boolean value of **True** or **False** to control the values for *accel_numerator* or *accel_denominator*.
do_threshold	Specifies a Boolean value of **True** or **False** to control the values for *accel_numerator* or *accel_denominator*.
accel_numerator	Specifies the numerator for the acceleration multiplier.
accel_denominator	Specifies the denominator for the acceleration multiplier.
threshold	Specifies the acceleration threshold.

XChangePointerControl defines how the pointing device moves. The acceleration, expressed as a fraction, is a multiplier for movement. For example, specifying **3/1** means the pointer moves three times as fast as normal. (The fraction may be rounded arbitrarily by the X Server.) Acceleration only takes effect if the pointer moves more than threshold pixels at once and applies to the amount beyond the value in the threshold argument only. To restore the default, set the value to **-1**.

The values of the *do_accel* and *do_threshold* arguments must be non-zero for the pointer values to be set. Otherwise, the parameters will not be changed.

XChangePointerControl can generate the event error **BadValue**.

¦ Copyright IBM Corp. 1985, 1991

2.4.76 XChangeProperty

XChangeProperty(*display, window, property, type, format, mode, data, nelements*
Display **display*;
Window *window*;
Atom *property*, *type*;
int *format*;
int *mode*;
unsigned char **data*;
int *nelements*;

display Specifies the connection to the X Server.

window Specifies the window ID for the window whose property is to be changed.

property Specifies the property atom.

type Specifies the type of the property.

format Specifies the format for the data as a list of 8-bit, 16-bit, or 32-bit quantities.

mode Specifies the mode.

data Specifies the property data.

nelements Specifies the number of elements of the specified data format in 8-bit, 16-bit, or 32-bit.

XChangeProperty changes the property for a specified window. (If the property does not exist, it adds the property.) It alters the property for the specified window and causes the X Server to generate a **PropertyNotify** event on that window. The property remains defined even after the client that defined it goes away. The property becomes undefined if the application calls **XDeleteProperty**, if the application destroys the specified window, or if the application closes the last connection to the X Server.

The *type* is not interpreted by the X Server. It is passed to an application program when the client calls **XGetProperty**.

The *format* allows the X Server to correctly perform byte-swap operations for the 8-bit, 16-bit, and 32-bit values. If *format* is 16-bit or 32-bit, it must explicitly cast the data pointer to a **(char *)** when it calls **XChangeProperty**.

The *mode* can be set to **PropModeReplace**, **PropModePrepend**, or **PropModeAppend**. **XChangeProperty** does the following according to the value assigned to this argument.

If *mode* is **PropModeReplace**, it discards the previous property value.

If *mode* is **PropModePrepend** or **PropModeAppend**, the type and format must match the existing property value. If the property is undefined, it is treated as defined with the correct type and format with zero-length data.

- For **PropModePrepend**, **XChangeProperty** inserts the data before the beginning of the existing data.

¦ Copyright IBM Corp. 1985, 1991

- For **PropModeAppend**, **XChangeProperty** appends the data onto the end of the existing data.

The lifetime of a property is not tied to the client. Properties remain until explicitly deleted, or the window is destroyed, or the server is reset.

XChangeProperty can generate the event errors **BadAlloc**, **BadAtom**, **BadMatch**, **BadValue**, and **BadWindow**.

¦ Copyright IBM Corp. 1985, 1991

2.4.77 XChangeSaveSet

XChangeSaveSet(*display, window_add, change_mode*)
Display **display*;
Window *window_add*;
int *change_mode*;

display Specifies the connection to the X Server.

window_add Specifies the window ID of the window whose children are to be added to the client's save-set.

change_mode Specifies the mode.

XChangeSaveSet adds or removes a subwindow from the client's save-set depending on the *change_mode* argument. The window specified must be created by another client.

The *change_mode* can be set to **SetModeInsert** or **SetModeDelete**.

If *change_mode* is **SetModeInsert**, **XChangeSaveSet** adds the window to the client save-set.

If *change_mode* is **SetModeDelete**, **XChangeSaveSet** deletes the window from the client save-set. The X Server automatically removes windows from the save-set when they are destroyed.

XChangeSaveSet can generate the event errors **BadMatch**, **BadValue**, and **BadWindow**.

¦ Copyright IBM Corp. 1985, 1991

2.4.78 XChangeWindowAttributes

XChangeWindowAttributes(*display, window, valuemask, attributes*)
Display **display*;
Window *window*;
unsigned long *valuemask*;
XSetWindowAttributes **attributes*;

display Specifies the connection to the X Server.

window Specifies the window ID.

valuemask Specifies the window attributes defined in the attributes argument. This mask is the bitwise inclusive OR of the valid attribute mask bits. If the *valuemask* is zero, the rest is ignored, and the attributes are not referenced.

attributes Specifies the attributes of the window that will be set at the time the specified window is created.

XChangeWindowAttributes uses the window attributes in the **XSetWindowAttributes** structure to change the specified window attributes depending on *valuemask*. See "Defining Window Attributes" in topic 1.5.3 for information about window attributes.

Changing the background does not cause the window contents to b changed. Use **XClearWindow** to repaint the window and the background of the window.
Setting the border, or changing the background such that the borde tile origin changes, causes the border to be repainted.
Changing the background of a root window to **None** or **ParentRelative** restores the default background pixmap.
Changing the border of a root window to **CopyFromParent** restores the default border pixmap.
Changing the *win_gravity* does not affect the current position of the window.
Changing the *backing_store* of an obscured window to **WhenMapped** or **Always** may have no immediate effect.
Changing the *backing_planes*, *backing_pixel*, or *save_under* of a mapped window may have no immediate effect.
Multiple clients can select input on the same window and the even masks are maintained separately. When an event is generated, it will be reported to all interested clients. However, only one client can select **SubstructureRedirectMask**, **ResizeRedirectMask**, and **ButtonPressMask** at a time.
There is only one *do_not_propagate_mask* per window, not one per client.
Changing the colormap of a window (that is, defining a new map, no changing the contents of the existing map) generates a **ColormapNotify** event.
Changing the colormap of a visible window may have no immediate effec on the screen because the colormap may not be installed. See **XInstallColormap** in this chapter.
Windows should share colormaps whenever possible
Changing the cursor of a root window to **None** restores the default cursor.

XChangeWindowAttributes can generate the event errors **BadAccess**, **BadColor**, **BadCursor**, **BadMatch**, **BadPixmap**, **BadValue**, and **BadWindow**.

¦ Copyright IBM Corp. 1985, 1991

2.4.79 XCheckIfEvent

Bool XCheckIfEvent(*display, event_return, predicate, arg*)
Display **display*;
XEvent **event_return*;
Bool (**predicate*)();
char **arg*;

display Specifies the connection to the X Server.

event_return Copies the matched event's associated structure into this client-supplied structure.

predicate Specifies the procedure to call to determine if the next event in the queue matches the one specified by the event argument.

arg Specifies the user-supplied argument to be passed to the predicate procedure.

XCheckIfEvent copies the matched event into the client-supplied **XEvent** structure when the predicate procedure finds a match. (See "Defining Event Structures" in topic 1.11.2.) **XCheckIfEvent** uses the following predicate procedure:

Bool predicate (*display, event, arg*)
Display **display*;
XEvent **event*;
char **arg*;

XCheckIfEvent returns **True** and removes this event from the queue if the matched event is found. If it does not find a match, it returns **False** and flushes the output buffer. Events stored in the queue earlier are not discarded.

¦ Copyright IBM Corp. 1985, 1991

2.4.80 XCheckMaskEvent

Bool XCheckMaskEvent(*display, event_mask, event_return*)
Display **display*;
unsigned long *event_mask*;
XEvent **event_return*;

display Specifies the connection to the X Server.

event_mask Specifies the event mask which is the bitwise inclusive OR of one or more of the valid event mask bits. See "Defining Event Masks" in topic 1.11.3.

event_return Copies the matched event's associated structure into this client-supplied structure.

XCheckMaskEvent searches first the event queue, then any events available on the server connection, for the first event that matches the specified mask. When it finds a match, it removes that event, copies it into the specified **XEvent** structure, and returns **True**. The other events stored in the queue are not discarded. See "Defining Event Structures" in topic 1.11.2 for information about the **XEvent** structure.

If the event requested is not in the queue, **XCheckMaskEvent** flushes the output buffer and returns **False**.

¦ Copyright IBM Corp. 1985, 1991

2.4.81 XCheckTypedEvent

int XCheckTypedEvent(*display, event_type, event_return*)
Display **display*;
int *event_type*;
XEvent **event_return*;

display Specifies the connection to the X Server.

event_type Specifies the event type to be compared. For a list of event types, see "Processing Events" in topic 1.11.4.

event_return Copies the matched event's associated structure into this client-supplied structure.

XCheckTypedEvent searches the event queue, then any events available on the server connection for the first event that matches the specified type. When it finds a match, it returns the associated event structure to the specified **XEvent** structure and returns **True**. The other events in the queue are not discarded. See "Defining Event Structures" in topic 1.11.2 for information about the **XEvent** data structure.

If the event is not available, **XCheckTypedEvent** flushes the output buffer and returns **False**.

¦ Copyright IBM Corp. 1985, 1991

2.4.82 XCheckTypedWindowEvent

int XCheckTypedWindowEvent(*display, window, event_type, event_return*)
Display **display*;
Window *window*;
int *event_type*;
XEvent **event_return*;

display Specifies the connection to the X Server.

window Specifies the window ID.

event_type Specifies the event type to be compared. For a list of event types, see "Processing Events" in topic 1.11.4.

event_return Copies the associated structure of the matched events into this client-supplied structure.

XCheckTypedWindowEvent searches the event queue, then any events available on the server connection for the first event that matches the specified type and window. When it finds a match, it removes the event from the queue, copies it into the specified **XEvent** structure and returns **True**. The other events in the queue are not discarded. See "Defining Event Structures" in topic 1.11.2 for information about the **XEvent** data structure.

If the event is not available, **XCheckTypedWindowEvent** flushes the output buffer and returns **False**.

For a list of valid event types, see "Processing Events" in topic 1.11.4.

¦ Copyright IBM Corp. 1985, 1991

2.4.83 XCheckWindowEvent

Bool XCheckWindowEvent(*display, window, event_mask, event_return*)
Display **display*;
Window *window*;
int *event_mask*;
XEvent **event_return*;

display Specifies the connection to the X Server.

window Specifies the window ID for the window whose next matched event you want to remove.

event_mask Specifies the event mask. This mask is the bitwise inclusive OR of one or more of the valid event mask bits. For a list of event masks, see "Defining Event Masks" in topic 1.11.3.

event_return Copies the matched event's associated structure into this client-supplied structure.

XCheckWindowEvent searches the event queue, then the events available on the server for the first event that matches the specified window and event mask. When it finds a match, it removes that event, copies it into the specified **XEvent** structure and returns **True**. The other events stored in the queue are not discarded. See "Defining Event Structures" in topic 1.11.2 for information about **XEvent**.

If the event you requested is not available, **XCheckWindowEvent** flushes the output buffer and returns **False**.

¦ Copyright IBM Corp. 1985, 1991

2.4.84 XCirculateSubwindows

XCirculateSubwindows(*display, window, direction*)
Display * *display*;
Window *window*;
int *direction*;

display Specifies the connection to the X Server.

window Specifies the window ID of the window to be circulated.

direction Specifies the direction for circulating the window.

XCirculateSubwindows circulates the specified subwindow in the direction specified. If another client has selected **SubstructureRedirectMask**, a **CirculateRequest** event is generated, and no further processing is performed. (See "Processing CirculateRequest Events" in topic 1.11.16.1.) Otherwise, the window is raised or lowered as specified.

If the window is restacked, the X Server generates a **CirculateNotify** event.

The *direction* can be **RaiseLowest** or **LowerHighest**.

If **RaiseLowest** is specified, **XCirculateSubwindows** raises the lowest mapped child (if any) that is occluded by another child to the top of the stack.
If **LowerHighest** is specified, **XCirculateSubwindows** lowers the highest mapped child (if any) that occludes another child to the bottom of the stack.

Exposure processing is performed on formerly obscured windows.

XCirculateSubwindows can generate the errors **BadValue** and **BadWindow**.

¦ Copyright IBM Corp. 1985, 1991

2.4.85 XCirculateSubwindowsDown

XCirculateSubwindowsDown(*display, window*)
Display **display*;
Window *window*;

display Specifies the connection to the X Server.

window Specifies the window ID for the window to be lowered.

XCirculateSubwindowsDown lowers the highest mapped child of the specified window that partially or completely occludes another child. Unobsured children are not affected. See "Configuring Windows" in topic 1.5.7.

XCirculateSubwindowsDown can generate the event error **BadWindow**.

¦ Copyright IBM Corp. 1985, 1991

2.4.86 XCirculateSubwindowsUp

XCirculateSubwindowsUp(*display, window*)
Display * *display*;
Window *window*;

display Specifies the connection to the X Server.

window Specifies the window ID of the window to be raised.

XCirculateSubwindowsUp raises the lowest mapped child of the specified window that is partially or completely occluded by another child. Unobscured children are not affected. See "Configuring Windows" in topic 1.5.7.

This routine can generate the event error **BadWindow**.

¦ Copyright IBM Corp. 1985, 1991

2.4.87 XClearArea

XClearArea(*display, window, x, y, width, height, exposures*)
Display **display*;
Window *window*;
int *x, y*;
unsigned int *width, height*;
Bool *exposures*;

display	Specifies the connection to the X server.
window	Specifies the window ID.
x, y	Specifies the *x* and *y* coordinates. These coordinates, which are relative to the origin of the window, specify the upper-left corner of the rectangle.
width, height	Specifies the *width* and *height* which are the dimensions of the rectangle to be cleared.
exposures	Specifies a Boolean value of **True** or **False**.

XClearArea paints a specified rectangular area in the specified window according to the specified dimensions with the background pixel or pixmap of the window.

If *width* is zero, it is replaced with the current width of the window minus *x*.
If *height* is zero, it is replaced with the current height of the window minus *y*.
If the window has a defined background tile, the rectangle is fille with this tile.
If the window has background **None**, the contents of the window are not changed.

In both cases, if *exposures* is **True**, one or more exposure events are generated for regions of the rectangle that are either visible or in backing store.

XClearArea cannot be used with an **InputOnly** class window.

This routine can generate the event errors **BadMatch**, **BadValue**, and **BadWindow**.

¦ Copyright IBM Corp. 1985, 1991

2.4.88 XClearWindow

XClearWindow(*display*, *window*)
Display **display*;
Window *window*;

display Specifies the connection to the X server.

window Specifies the window ID of the window to be cleared.

XClearWindow clears the entire area in the window specified.

If the window has a defined background tile, the rectangle is tile with a plane mask of ones and the function **GXcopy**.
If the window background is **None**, the contents of the window are not changed.

XClearWindow cannot be used with an **InputOnly** class window.

XClearWindow can generate the event errors **BadMatch**, **BadValue**, and **BadWindow**.

¦ Copyright IBM Corp. 1985, 1991

2.4.89 XClipBox

XClipBox(*region, rect*)
Region *region*;
XRectangle **rect*;

region Specifies the region where the rectangle is located.

rect Specifies the rectangle in which the smallest enclosing rectangle is generated.

XClipBox generates the smallest enclosing rectangle in *rect*. The opaque type **Region** is defined in **<X11/Xutil.h>**.

See "Drawing Points, Lines, Rectangles, and Arcs" in topic 1.8.3 for a description of the **XRectangle** data structure.

¦ Copyright IBM Corp. 1985, 1991

2.4.90 XCloseDisplay

XCloseDisplay(*display*)
Display **display*;

display Specifies the connection to the X Server.

XCloseDisplay closes or disconnects a display from the X Server. It destroys all windows, resource IDs (**Window, Font, Pixmap, Colormap, Cursor**, and **GContext**), or other **GC**s that the client created on the display, unless the close-down mode of the resource is changed. It also discards any output requests that were buffered but not sent.

These windows, resource IDs, and other **GC**s should not be referenced again.

¦ Copyright IBM Corp. 1985, 1991

2.4.91 XConfigureWindow

XConfigureWindow(*display, window, value_mask, values*)
Display **display*;
Window *window*;
unsigned int *value_mask*;
XWindowChanges **values*;

display Specifies the connection to the X Server.

window Specifies the window ID of the window to be reconfigured.

value_mask Specifies the values to be set in the *values* variable. This mask is the bitwise inclusive OR of the valid change window values bits.

values Specifies a pointer to the structure **XWindowChanges**.

XConfigureWindow reconfigures the size, position, border and stacking order of a window. It uses the values specified in the **XWindowChanges** structure. Otherwise, it takes the values from the existing geometry of the window. The stacking order of the window is controlled by the arguments in the function. See "Configuring Windows" in topic 1.5.7 for information on the value masks and the **XWindowChanges** data structure.

XConfigureWindow can generate the event errors **BadMatch**, **BadValue**, and **BadWindow**.

¦ Copyright IBM Corp. 1985, 1991

2.4.92 XConvertSelection

XConvertSelection(*display, selection, target, property, requestor, time*)
Display **display*;
Atom *selection, target*;
Atom *property*;
Window *requestor*;
Time *time*;

display Specifies the connection to the X Server.

selection Specifies the selection atom.

target Specifies the target atom.

property Specifies the property atom.

requestor Specifies the window ID of the window initiating the request.

time Specifies the time in a timestamp, which is expressed in milliseconds, or **CurrentTime**.

XConvertSelection requests a selection conversion. It requests that the specified selection be converted to the specified target type:

If the specified selection has an owner, the X Server sends **SelectionRequest** event to that owner. For more information, see "Processing SelectionRequest Events" in topic 1.11.18.4.

If the specified selection does not have an owner, the X Serve generates a **SelectionNotify** event to the requestor with property **None**. For more information, see "Processing SelectionNotify Events" in topic 1.11.18.5.

In both events, the arguments are passed unchanged.

XConvertSelection can generate the event errors **BadAtom** and **BadWindow**.

¦ Copyright IBM Corp. 1985, 1991

2.4.93 XCopyArea

XCopyArea(*display, src, dest, gc, src_x, src_y, width, height, dest_x, dest_y*)
Display **display*;
Drawable *src, dest*;
GC *gc*;
int *src_x, src_y*;
unsigned int *width, height*;
int *dest_x, dest_y*;

display	Specifies the connection to the X server.
src, dest	Specifies the source and destination rectangles to be combined.
gc	Specifies the graphics context.
src_x, src_y	Specifies the *x* and *y* coordinates of the source rectangle relative to its origin. These coordinates specify the upper-left corner of the source rectangle.
width, height	Specifies the *width* and *height* of both the source and destination rectangles.
dest_x, dest_y	Specifies the *x* and *y* coordinates of the destination rectangle relative to its origin. These coordinates specify the upper-left corner of the destination rectangle.

XCopyArea copies an area of the specified drawable between drawables. It combines the source (*src*) rectangle with the destination (*dest*) rectangle. The rectangles specified by these two arguments must have the same root and depth.

If the regions of the source rectangle are obscured and have not been retained in *backing_store*, these regions are not copied. If the regions outside the boundaries of the source drawable are specified, those regions are not copied. Instead, the following occurs on all corresponding destination regions that are either visible or retained in *backing_store*.

> If the destination rectangle is a window with a background other tha **None**, these corresponding regions of the destination are tiled with *plane_mask* of all ones and function **GXcopy** with that background.
>
> If *graphics_exposures* in **GC** is **True**, then **GraphicsExpose** events for all corresponding destination regions are generated.
>
> If *graphics_exposures* is **True**, but no regions are exposed, a **NoExpose** event is generated. By default, *graphics_exposures* is **True** in new **GC**s. See "Using Events and Event-Handling Functions" in topic 1.11 for further information.

XCopyArea uses the graphics context components: *function, plane_mask, subwindow_mode, graphics_exposures, clip_x_origin, clip_y_origin*, and *clip_mask*. See "Manipulating Graphics Context or State" in topic 1.7.8.

This routine can generate the event errors **BadDrawable**, **BadGC**, and **BadMatch**.

¦ Copyright IBM Corp. 1985, 1991

2.4.94 XCopyColormapAndFree

Colormap XCopyColormapAndFree(*display, cmap*)
Display **display*;
Colormap *cmap*;

display Specifies the connection to the X Server.

cmap Specifies the colormap ID.

XCopyColormapAndFree creates a new colormap when allocating from a previously-shared colormap that has failed because of resource exhaustion. **XCopyColormapAndFree** does the following:

Creates a colormap of the same visual type and for the same screen a the *cmap* argument and returns the new colormap ID.

Moves all of the client's existing allocation from *cmap* to the new colormap with the color values intact. It also frees the color entries in the colormap. Color values in other entries in the new colormap are undefined.

If *cmap* was created by the client with the *alloc* argument set to **AllocAll**, the new colormap is also created with **AllocAll**. The color values for all entries are copied from *cmap*, and then all entries in *cmap* are freed.

If *cmap* was not created by clients with **AllocAll**, pixels and planes allocated by the client using **XAllocColor**, **XAllocNamedColor**, **XAllocColorCells**, or **XAllocColorPlanes** are moved.

XCopyColormapAndFree can generate the event errors **BadAlloc** and **BadColor**.

¦ Copyright IBM Corp. 1985, 1991

2.4.95 XCopyGC

XCopyGC(*display, src, valuemask_copy, dest*)
Display **display*;
GC *src*;
unsigned long *valuemask_copy*;
GC *dest*;

display	Specifies the connection to the X Server.
src	Specifies the components of the source graphics context.
valuemask_copy	Specifies the components in the source graphics context to be copied to the destination graphics context. This argument is the bitwise inclusive OR of one or more of the valid **GC** component masks.
dest	Specifies the destination graphics context.

XCopyGC copies components from a source (*src*) graphics context to a destination (*dest*) graphics context. Both graphics contexts must have the same root and depth. For more information about valid **GC** component masks, see "Manipulating Graphics Context or State" in topic 1.7.8.

XCopyGC can generate the event errors **BadAlloc**, **BadGC**, **BadMatch**, and **BadValue**.

¦ Copyright IBM Corp. 1985, 1991

2.4.96 XCopyPlane

XCopyPlane(*display, src, dest, gc, src_x, src_y, width, height, dest_x, dest_y, plane*)
Display **display*;
Drawable *src, dest*;
GC *gc*;
int *src_x, src_y*;
unsigned int *width, height*;
int *dest_x, dest_y*;
unsigned long *plane*;

display Specifies the connection to the X server.

src, dest Specifies the source and destination rectangles to be combined.

gc Specifies the graphics context.

src_x, src_y Specifies the *x* and *y* coordinates of the source rectangle relative to its origin. These coordinates specify the upper-left corner of the source rectangle.

width, height Specifies the *width* and *height* of both the source and destination rectangles.

dest_x, dest_y Specifies the *x* and *y* coordinates of the destination rectangle relative to its origin. These coordinates specify the upper-left corner of the destination rectangle.

plane Specifies the bit-plane. Set one bit, for example:

0x01, 0x02, 0x04 (**0x03** is illegal.)

XCopyPlane combines a one-bit plane of the source (*src*) rectangle with the destination (*dest*) rectangle. The rectangles must have the same root, but not necessarily the same depth.

XCopyPlane forms a pixmap of the same depth as the destination rectangle with a size specified by the source region. It uses the foreground pixels in the graphics context where the bit-plane in the source rectangle contains a one-bit; and the background pixels in the graphics context where the bit-plane in the source rectangle contains a zero bit.

The equivalent of an **XCopyArea** request is performed with all the same exposure semantics. This can be thought of as using the specified region of the source bit-plane as a stipple with a *fill_style* of **FillOpaqueStippled** for filling a rectangular area of the destination. See *line_style* in "Manipulating Graphics Context or State" in topic 1.7.8.

If *graphics_exposures* in **GC** is **True**, the **GraphicsExpose** events for all corresponding destination regions are generated. If *graphics_exposures* is **True**, but no regions are exposed, a **NoExpose** event is generated. By default, *graphics_exposures* is **True** in new **GC**s. See "Using Events and Event-Handling Functions" in topic 1.11.

XCopyPlane uses the graphics context components *function, plane_mask, foreground, background, subwindow_mode, graphics_exposures, clip_x_origin, clip_y_origin*, and *clip_mask*. See "Manipulating Graphics Context or State"

¦ Copyright IBM Corp. 1985, 1991

in topic 1.7.8.

XCopyPlane can generate the event errors **BadDrawable**, **BadGC**, **BadMatch**, and **BadValue**.

¦ Copyright IBM Corp. 1985, 1991

2.4.97 XCreateBitmapFromData

Pixmap XCreateBitmapFromData(*display, drawable, data, width, height*)
Display **display*;
Drawable *drawable*;
char **data*;
int *width, height*;

display Specifies the connection to the X Server.

drawable Specifies the drawable.

data Specifies the location of the bitmap data.

width, height Specifies the *width* and *height* of the bitmap to be created.

XCreateBitmapFromData allows you to include, in your **C** language program using **#include**, a bitmap file that was written out by **XWriteBitmapFile** without reading in the bitmap file. This function works on the AIX RT X-Windows Version 2.1 format only. The following is an example for including a gray bitmap:

```
#include "gray.bitmap"
userpixmap = XCreateBitmapFromData(display, window, gray_bits,
                        gray_width, gray_height);
```

If insufficient working storage was allocated, **XCreateBitmapFromData** returns **NULL**.

Free the bitmap with **XFreePixmap** after this function is completed.

¦ Copyright IBM Corp. 1985, 1991

2.4.98 XCreateColormap

Colormap XCreateColormap(*display, window, visual, alloc*)
Display **display*;
Window *window*;
Visual **visual*;
int *alloc*;

display Specifies the connection to the X Server.

window Specifies the window ID for the screen where the colormap is to be created.

visual Specifies a pointer to a visual type supported on the screen.

alloc Specifies the colormap entries to be allocated.

The **XCreateColormap** function creates a colormap of the specified visual type for the screen on which the window resides and associates the colormap ID with it. **XCreateColormap** operates on a **Visual** structure whose members contain information about the colormapping possible. This function does not set the colormap for the specified window. **XCreateColormap** determines the correct screen only.

The members of the **Visual** structure pertinent to **XCreateColormap** include the following:

The *class* member specifies the screen class. It can be set to one of the following: **GrayScale, PseudoColor, DirectColor, StaticColor, StaticGray**, or **TrueColor**.

The *red_mask*, *green_mask*, and *blue_mask* members specify the color mask values.

The *map_entries* member specifies the number of colormap entries. The initial values for *map_entries* are defined according to the *class* member.

- If *class* is **GrayScale, PseudoColor**, or **DirectColor**, the initial values for *map_entries* are undefined.

- If *class* is **StaticColor**, **StaticGray**, or **TrueColor**, the initial values for *map_entries* are defined. These values are specific to the visual type and are not defined by the X Server.

The *alloc* member specifies the colormap entries to be allocated. The *class* member determines which constant is passed to the *alloc* argument:

- If *class* is **StaticGray**, **StaticColor**, or **TrueColor**, *alloc* should be set to **AllocNone**.

- If *class* is **GrayScale, PseudoColor**, or **DirectColor**, the *alloc* should be set to **AllocNone**. In this case, the colormap has no defined values for *map_entries*. This allows your client programs to allocate the entries in the colormap. It can also be set to **AllocAll**. In this case, **XCreateColormap** allocates the entire colormap as writable. The initial values of all *map_entries* are undefined and cannot be freed with a call to **XFreeColors**.

¦ Copyright IBM Corp. 1985, 1991

When *class* is **GrayScale** or **PseudoColor**, the processing simulates a call to **XAllocColor**, where **XAllocColor** returns all pixel values from 0 to N - 1. The value N represents the *map_entries* value in the **Visual** structure specified.

When *class* is **DirectColor**, the processing simulates a call to **XAllocColorPlanes**, where **XAllocColorPlanes** returns pixel values of zero and *rmask*, *gmask*, and *bmask* values that contain the same bits as the *red_mask*, *green_mask*, and *blue_mask* members in the **Visual** structure specified.

XCreateColormap can generate the event errors **BadAlloc**, **BadMatch**, **BadValue**, and **BadWindow**.

¦ Copyright IBM Corp. 1985, 1991

2.4.99 XCreateFontCursor

```
#include <X11/cursorfont.h>

Cursor XCreateFontCursor(display, shape)
Display *display;
unsigned int shape;
```

display Specifies the connection to the X Server.

shape Specifies the shape for the cursor.

XCreateFontCursor creates a cursor from a standard font. A set of standard cursor shapes is available in a special font named **cursorfont**. This interface helps customize fonts for individual display types. See Appendix B, "Xlib Cursor Fonts" in topic B.0 for a list of cursors and font shapes.

The *shape* argument specifies which **glyph** of the standard fonts to use. The hotspot comes from the information stored in the cursor font. The initial colors of a cursor are a black foreground and a white background.

This routine can generate the event errors **BadAlloc**, **BadMatch**, and **BadValue**.

¦ Copyright IBM Corp. 1985, 1991

2.4.100 XCreateGC

GC XCreateGC(*display, drawable, valuemask_create, values*)
Display **display*;
Drawable *drawable*;
unsigned long *valuemask_create*;
XGCValues **values*;

display	Specifies the connection to the X Server.
drawable	Specifies the drawable.
valuemask_create	Specifies the components in the graphics context being created to be set using the information in the **XGCValues** structure. This argument is the bitwise inclusive OR of one or more of the valid values GC component masks.
values	Specifies a pointer to the **XGCValues** structure.

XCreateGC creates a new graphic context that can be used with any destination drawable with the same root and depth as the specified drawable. For information on the **XGCValues** data structure and valid **GC** components, see "Manipulating Graphics Context or State" in topic 1.7.8.

XCreateGC can generate the event errors **BadAlloc**, **BadDrawable**, **BadFont**, **BadPixmap**, **BadMatch**, and **BadValue**.

¦ Copyright IBM Corp. 1985, 1991

2.4.101 XCreateGlyphCursor

Cursor XCreateGlyphCursor(*display, source_font, mask_font, source_char,*
*mask_char, foreground_color, background_*
Display **display*;
Font *source_font, mask_font*;
unsigned int *source_char, mask_char*;
XColor **foreground_color*;
XColor **background_color*;

display Specifies the connection to the X Server.

source_font Specifies the font for the source glyph.

mask_font Specifies the font for the mask glyph.

source_char Specifies the character glyph for the source.

mask_char Specifies the character glyph for the mask.

foreground_color Specifies the red, green, and blue (RGB) values for the foreground of the source. See "Creating Colormaps" in topic 1.7.1.

background_color Specifies the red, green, and blue (RGB) values for the background of the source. See "Creating Colormaps" in topic 1.7.1.

XCreateGlyphCursor creates a cursor from font glyphs. The source and mask bitmaps are obtained from the specified font glyphs.

The *source_char* must be a defined glyph in *source_font*.
If *mask_font* is specified, *mask_char* must be a defined glyph in *mask_font*. The *mask_font* and *mask_char* are optional.
If defined, the origins of the *source_char* and *mask_char* glyphs are positioned coincidently and define the hotspot. The *source_char* and *mask_char* do not need to have the same bounding box metrics. There is no restriction on the placement of the hotspot relative to the bounding boxes. If no *mask_char* is given, all pixels of the source are displayed.
The *source_char* and *mask_char* are of type **unsigned int**. For 2-byte matrix fonts, the 16-bit value should be formed with the *byte1* member in the most-significant byte and the *byte2* member in the least-significant byte.

Use **XFreeFont** to free the fonts immediately if no further explicit references to them are to be made.

XCreateGlyphCursor can generate the event errors **BadAlloc**, **BadFont**, and **BadValue**.

¦ Copyright IBM Corp. 1985, 1991

2.4.102 XCreateImage

XImage *XCreateImage(*display, visual, depth, format, offset, data, width, height, bitmap_pad, bytes_per_line*
Display **display*;
Visual **visual*;
unsigned int *depth*;
int *format*;
int *offset*;
char **data*;
unsigned int *width*;
unsigned int *height*;
int *bitmap_pad*;
int *bytes_per_line*;

display	Specifies the connection to the X Server.
visual	Specifies a pointer to the visual.
depth	Specifies the depth of the image.
format	Specifies the format (**XYPixmap** or **ZPixmap**) for the image.
offset	Specifies the offset (in pixels) to ignore at the beginning of the scanline permitting the rapid display of the image without requiring each scanline to be shifted into position.
data	Specifies a pointer to the image data.
width, height	Specifies the *width* and *height* (in pixels) of the image.
bitmap_pad	Specifies the quantum of a scanline.
bytes_per_line	Specifies the number of bytes in the client image between the start of one scanline and the start of the next.

XCreateImage allocates the memory for an **XImage** structure for the specified display but does not allocate space for the image itself. Rather, it initializes the structure with default values and returns a pointer to the **XImage** structure. See "Transferring Images Between Client and Server" in topic 1.9.3 for information about the **XImage** data structure.

The red, green, and blue mask values from the **Visual** structure are defined for Z format images only. See "Defining Visual Types" in topic 1.5.1 for more information.

The *bitmap_pad* indicates the start of one scanline which is separated in client memory from the start of the next scanline by an integer multiple of the number of bits (8-bit, 16-bit, or 32-bit) indicated in this variable.

If the *bytes_per_line* is set to zero, **Xlib** assumes that the scanlines are contiguous in memory and calculates the value of *bytes_per_line* itself.

The basic functions used to get a pixel, set a pixel, create a subimage, and add a constant offset to a Z format image are defined in the image object. The macros to call through the image object are defined in the **<**

¦ Copyright IBM Corp. 1985, 1991

X11/Xutil.h > file.

To get your own routines for **XGetPixel**, **XSetPixel**, **XSubImage**, and **XAddPixel**, change the definitions in the **< X11/Xutil.h >** file.

¦ Copyright IBM Corp. 1985, 1991

2.4.103 XCreatePixmap

Pixmap XCreatePixmap(*display, drawable, width, height, depth*)
Display **display*;
Drawable *drawable*;
unsigned int *width, height*;
unsigned int *depth*;

display	Specifies the connection to the X Server.
drawable	Specifies the screen on which to create the pixmap.
width, height	Specifies the *width* and *height* of the pixmap. This value must be a non-zero.
depth	Specifies the depth of the pixmap which must be supported by the root of the specified drawable.

XCreatePixmap creates a pixmap of a specified size. It assigns the pixmap ID to the created pixmap. This function can be used with an **InputOnly** window as the *drawable* argument.

The server uses the *drawable* argument to determine on which screen the pixmap is stored. The pixmap can be used only on this screen and only with other drawables of the same depth. (2) (See **XCopyPlane** for an exception to this rule.)

The initial contents of the pixmap are undefined. If **XCreatePixmap** returns a zero, there is insufficient space for the pixmap.

XCreatePixmap can generate the event errors **BadAlloc**, **BadDrawable**, and **BadValue**.

(2) AIX X-Windows supports only one screen.

¦ Copyright IBM Corp. 1985, 1991

2.4.104 XCreatePixmapCursor

Cursor XCreatePixmapCursor(*display, source, mask, foreground_color, background_color, x, y*)
Display **display*;
Pixmap *source*;
Pixmap *mask*;
XColor **foreground_color*;
XColor **background_color*;
unsigned int *x, y*;

display	Specifies the connection to the X Server.
source	Specifies the shape of the source cursor.
mask	Specifies the source bits of the cursor that are to be displayed.
foreground_color	Specifies the red, green, and blue (RGB) values for the foreground of the source. See "Creating Colormaps" in topic 1.7.1.
background_color	Specifies the red, green, and blue (RGB) values for the background of the source. See "Creating Colormaps" in topic 1.7.1.
x, y	Specifies the *x* and *y* coordinates which indicate the hotspot relative to the origin of the source. These coordinates must be a point within the source cursor.

XCreatePixmapCursor creates a cursor and returns the cursor ID associated with it.

Specify the **RGB** values for the *foreground_color* and *background_color*, even if the X Server has a **StaticGray** or **GrayScale** screen only.

The foreground color is used for the one bits in the source, and the background color is used for the zero bits.

Both *source* and *mask* arguments, if specified, must have depth of one with any root.

The *mask* argument defines the shape of the cursor; the one bits in the mask define the source pixels to be displayed. The corresponding bits of the source pixmap are ignored if the mask has zero bits. If no mask is given, all source pixels are displayed. The mask, if present, must be the same size as the pixmap defined in the *source* argument.

The components of the cursor may be transformed arbitrarily to meet display limitations. The pixmaps can be freed immediately if no further explicit references to them are to be made. Subsequent drawing in the source or mask pixmap is undefined on the cursor.

XCreatePixmapCursor can generate the event errors **BadAlloc**, **BadMatch**, and **BadPixmap**.

¦ Copyright IBM Corp. 1985, 1991

2.4.105 XCreatePixmapFromBitmapData

Pixmap XCreatePixmapFromBitmapData(*display, drawable, data, width, height,*
foreground, backgrounc
Display **display*;
Drawable *drawable*;
char **data*;
unsigned int *width, height*;
unsigned long *foreground, background*;
unsigned int *depth*;

display	Specifies the connection to the X Server.
data	Specifies the drawable.
width, height	Specifies the *width* and *height* for the pixmap to be created.
foreground, background	Specifies the foreground and background pixel values to be used.
depth	Specifies the depth of the pixmap.

XCreatePixmapFromBitmapData creates the pixmap of the depth specified and performs an **XPutImage** of the data into the pixmap. Clients can perform this routine manually.

¦ Copyright IBM Corp. 1985, 1991

2.4.106 XCreateRegion

Region XCreateRegion()

XCreateRegion creates a new empty region.

¦ Copyright IBM Corp. 1985, 1991

2.4.107 XCreateSimpleWindow

Window XCreateSimpleWindow(*display, parent, x, y, width, height, borderwidth, border, background*)
Display **display*;
Window *parent*;
int *x, y*;
unsigned int *width, height, borderwidth*;
unsigned long *border*;
unsigned long *background*;

display Specifies the connection to the X Server.

parent Specifies the parent window ID.

x, y Specifies the *x* and *y* coordinates. These coordinates, which are relative to the inside of the border of the parent window, define the top-left outside corner of the border of the created window.

width, height Specifies the *width* and *height* which are the dimensions of the inside of the created window, excluding the border. The *width* and *height* must be non-zero.

borderwidth Specifies the width (in pixels) of the border of the created window. The *borderwidth* for an **InputOnly** window must be zero.

border Specifies the border pixel of the window.

background Specifies the pixel value to be set for the background of the created window.

XCreateSimpleWindow creates an unmapped **InputOutput** subwindow for a specified parent window. It returns the window ID of the created window, and causes the X Server to generate a **CreateNotify** event.

The created window is placed on top in the stacking order with respect to siblings. Any part of the window that extends outside its parent window will be clipped.

The created window inherits the depth, class, and visual attributes from the parent window. Other window attributes use the default values. See "Creating Windows" in topic 1.5.4.

For the created window to be visible on the screen, the window and all of its ancestors must be mapped and it cannot be obscured by any of its ancestors. Then, the created window can be displayed by calling **XMapWindow**.

Initially, the created window will have the same cursor as the parent window. The cursor for the created window will be **None**. Use **XDefineCursor** to define a new cursor for the created window.

XCreateSimpleWindow can generate event errors **BadAlloc**, **BadMatch**, **BadValue**, and **BadWindow**.

¦ Copyright IBM Corp. 1985, 1991

2.4.108 XCreateWindow

Window XCreateWindow(*display, parent, x, y, width, height, borderwidth, depth, class, visual, valuemask, attributes*
Display **display*;
Window *parent*;
int *x, y*;
unsigned int *width, height*;
unsigned int *borderwidth*;
int *depth*;
unsigned int *class*;
Visual **visual*;
unsigned long *valuemask*;
XSetWindowAttributes **attributes*;

display Specifies the connection to the X Server.

parent Specifies the parent window ID.

x, y Specifies the *x* and *y* coordinates, which are relative to the inside of the borders of the parent window, define the top-left, outside corner of the border.

width, height Specifies the *width* and *height*, which are the dimensions of the inside of the created window. These dimensions do not include the border of the created window, which is entirely outside of the window. The *width* and *height* must be non-zero.

borderwidth Specifies the width of the border of the new window in pixels.

depth Specifies the depth of the new window.

class Specifies the class of the created window as **InputOutput**, **InputOnly**, or **CopyFromParent**.

visual Specifies the visual type.

valuemask Specifies the window attributes to be set in the attributes argument. This mask is the bitwise inclusive OR of the valid attribute mask bits. If *valuemask* is zero, the rest is ignored and the attributes are not referenced.

attributes Specifies the attributes of the window to be set when the window is created. The *valuemask* should have the appropriate bits set to indicate which attributes in the structure were set. See "Defining Window Attributes" in topic 1.5.3.

XCreateWindow creates an unmapped subwindow for a specified parent window. It returns the window ID of the created window, and it causes the X Server to generate a **CreateNotify** event. The created window is placed on top in the stacking order with respect to siblings. For more information, see "Creating Windows" in topic 1.5.4.

For an **InputOutput** window, the visual type and depth must be a combination supported for the screen. The depth does not need to be the same as the parent, but the parent window cannot be an **InputOnly** window.

¦ Copyright IBM Corp. 1985, 1991

For an **InputOnly** window, the depth must be zero and the visual must be supported by the screen. The parent window, however, can have any depth and class. The *borderwidth* for an **InputOnly** window must be zero.

The only window attributes defined for **InputOnly** windows are *win_gravity*, *event_mask*, *do_not_propagate_mask*, *override_redirect*, and *cursor*.

If *class* is **CopyFromParent**, the depth is taken from the parent window. If the *visual* is **CopyFromParent**, the visual type is taken from the parent window.

XCreateWindow can generate event errors **BadAlloc**, **BadColor**, **BadCursor**, **BadMatch**, **BadPixmap**, **BadValue**, and **BadWindow**.

¦ Copyright IBM Corp. 1985, 1991

2.4.109 XDefineCursor

XDefineCursor(*display, window, cursor*)
Display **display*;
Window *window*;
Cursor *cursor*;

display Specifies the connection to the X Server.

window Specifies the window ID.

cursor Specifies the cursor to be displayed when the pointer is in the specified window. If no cursor is to be displayed, it passes **None**.

XDefineCursor defines which cursor will be used in a window. If a cursor is set, it will be used when the pointer is in the window.

This routine can generate the event errors **BadAlloc**, **BadCursor**, and **BadWindow**.

¦ Copyright IBM Corp. 1985, 1991

2.4.110 XDeleteContext

int XDeleteContext(*display, window, context*)
Display **display*;
Window *window*;
XContext *context*;

display Specifies the connection to the X Server.

window Specifies the window ID of the window with which the data is associated.

context Specifies the context type to which the data belongs.

XDeleteContext deletes the entry for the given window and type from the data structure. This function returns a zero if it is successful. It returns a non-zero if it is not successful.

XDeleteContext can generate the error **XCNOENT**.

¦ Copyright IBM Corp. 1985, 1991

2.4.111 XDeleteModifiermapEntry

XModifierKeymap *__XDeleteModifiermapEntry__(*modmap, keycode, modifier*)*
XModifierKeymap **modmap*;
KeyCode *keycode*;
int *modifier*;

modmap Specifies a pointer to the **XModifierKeymap** structure.

keycode Specifies the keycode to be deleted.

modifier Specifies the modifier.

XDeleteModifiermapEntry deletes the specified keycode from the set that controls the specified modifier. It returns the result to the **XModifierKeymap** structure. For information about the **XModifierKeymap** data structure, see "Manipulating Keyboard Encoding" in topic 1.10.5.

¦ Copyright IBM Corp. 1985, 1991

2.4.112 XDeleteProperty

XDeleteProperty(*display, window, property*)
Display **display*;
Window *window*;
Atom *property*;

display Specifies the connection to the X Server.

window Specifies the window ID for the window whose property is to be deleted.

property Specifies the property atom.

XDeleteProperty deletes a property for the specified window only if the property was defined on the specified window. The X Server generates a **PropertyNotify** event on the window, unless the property does not exist. (See "Processing PropertyNotify Events" in topic 1.11.18.2.)

XDeleteProperty can generate the event errors **BadAtom** and **BadWindow**.

¦ Copyright IBM Corp. 1985, 1991

2.4.113 XDestroyImage

int XDestroyImage(*ximage*)
XImage **ximage;*

ximage Specifies a pointer to the image.

XDestroyImage deallocates the memory allocated in a previous call to **XCreateImage**. It deallocates the memory associated with the **XImage** structure. For information about the **XImage** data structure, see "Transferring Images Between Client and Server" in topic 1.9.3.

When the image is created with **XCreateImage** or **XGetImage**, this procedure usually frees both the image structure and the data pointed to by that image structure.

¦ Copyright IBM Corp. 1985, 1991

2.4.114 XDestroyRegion

XDestroyRegion(*region*)
Region *region;*

region Specifies the region.

XDestroyRegion deallocates the storage associated with a specified region.

¦ Copyright IBM Corp. 1985, 1991

2.4.115 XDestroySubwindows

XDestroySubwindows(*display, window*)
Display **display*;
Window *window*;

display Specifies the connection to the X Server.

window Specifies the window ID of the window to be destroyed.

XDestroySubwindows destroys all inferior windows of the specified window in a bottom-to-top stacking order. The X Server generates a **DestroyNotify** event for each window.

If any mapped subwindows are destroyed, the X Server generates exposure events on the specified window. The subwindows should not be referenced again.

By default, windows are destroyed when a connection is closed.

XDestroySubwindows can generate the event error **BadWindow**.

¦ Copyright IBM Corp. 1985, 1991

2.4.116 XDestroyWindow

XDestroyWindow(*display, window*)
Display **display*;
Window *window*;

display Specifies the connection to the X Server.

window Specifies the window ID of the window to be destroyed.

XDestroyWindow destroys the specified window and its subwindows. The X Server generates a **DestroyNotify** event for each window. The window should not be referenced again. (If the root window is the window specified, no windows are destroyed.)

If the window specified is a mapped window, **XDestroyWindow** unmaps it automatically. Then, the specified window and all its inferiors are destroyed. The X Server generates a **DestroyNotify** event for each window.

A **DestroyNotify** event is generated on the inferior windows before it is generated on the specified window. The ordering among siblings and across subhierarchies is not constrained, otherwise.

XDestroyWindow can generate the event error **BadWindow**.

¦ Copyright IBM Corp. 1985, 1991

2.4.117 XDisableAccessControl

XDisableAccessControl(*display*)
Display **display*;

display Specifies the connection to the X Server.

XDisableAccessControl disables the use of the access control list at connection setups. See "Controlling Host Access" in topic 1.10.6.

For this function to execute successfully, the client application must reside on the same host as the X Server and have the required permission in the initial authorization at connection setup.

This routine can generate the event error **BadAccess**.

¦ Copyright IBM Corp. 1985, 1991

2.4.118 XDisplayKeycodes

XDisplayKeycodes(*display, min_keycodes_return, max_keycodes_return*)
Display **display*;
int **min_keycodes_return*;
int **max_keycodes_return*;

display	Specifies the connection to the X Server.
min_keycodes_return	Returns the minimum number of KeyCodes.
max_keycodes_return	Returns the maximum number of KeyCodes.

XDisplayKeycodes returns the minimum and the maximum number of KeyCodes supported by the specified display. The *min_keycodes_return* is is never less than 8. The *max_keycodes_return* is never more than 255. Not all the KeyCodes in this range must have corresponding keys.

¦ Copyright IBM Corp. 1985, 1991

2.4.119 XDisplayMotionBufferSize

unsigned long XDisplayMotionBufferSize(*display*)
Display **display*;

display Specifies the connection to the X Server.

XDisplayMotionBufferSize returns the size of the motion buffer. It retains the recent history of the pointer motion. It is done so to a finer granularity than is reported by **MotionNotify** events. **XGetMotionEvents** makes this history available.

¦ Copyright IBM Corp. 1985, 1991

2.4.120 XDisplayName

char *__XDisplayName__(*string*)
char **string*;

string Specifies the character string.

XDisplayName reports an error to the user when the display requested does not exist. It returns the name of the display currently being used.

If a **NULL** string is specified, **XDisplayName** looks in the environment for the display and returns the name of the displayed requested.

¦ Copyright IBM Corp. 1985, 1991

2.4.121 XDrawArc

XDrawArc(*display, drawable, gc, x, y, width, height, angle1, angle2*)
Display **display*;
Drawable *drawable*;
GC *gc*;
int *x*, *y*;
unsigned int *width*, *height*;
int *angle1*, *angle2*;

display	Specifies the connection to the X Server.
drawable	Specifies the drawable.
gc	Specifies the graphics context.
x, y	Specifies the *x* and *y* coordinates.
width, height	Specifies *width* and *height* which are the major and minor axes of the arc.
angle1	Specifies the start of the arc relative to the 3 o'clock position from the center in units of degrees multiplied by 64.
angle2	Specifies the path and extent of the arc relative to the start of the arc in units of degrees multiplied by 64.

XDrawArc draws a single circular or elliptical arc in the specified drawable. Each arc is specificed by a rectangle and two angles. See "Drawing Single and Multiple Arcs" in topic 1.8.3.4.

This function uses the graphics context components: *function, plane_mask, line_width, line_style, cap_style, join_style, fill_style, subwindow_mode, clip_x_origin, clip_y_origin*, and *clip_mask*. It also uses the graphics context mode-dependent components: *foreground, background, tile, stipple, ts_x_origin, ts_y_origin, dash_offset*, and *dash_list*. See "Manipulating Graphics Context or State" in topic 1.7.8.

XDrawArc can generate the event errors **BadDrawable**, **BadGC**, and **BadMatch**.

¦ Copyright IBM Corp. 1985, 1991

2.4.122 XDrawArcs

XDrawArcs(*display, drawable, gc, arcs, narcs*)
Display **display*;
Drawable *drawable*;
GC *gc*;
XArc **arcs*;
int *narcs*;

display Specifies the connection to the X Server.

drawable Specifies the drawable.

gc Specifies the graphics context.

arcs Specifies a pointer to an array of arcs.

narcs Specifies the number of arcs in the array.

XDrawArcs draws multiple circular or elliptical arcs in the specified drawable. For this function, an array of **XArc** structures is specified. Each structure contains the x, y coordinates, width and height, start position and path and extent. See "Drawing Single and Multiple Arcs" in topic 1.8.3.4 for more information.

XDrawArcs draws the arcs in the order listed in the array. No pixel is drawn more than once if the following occurs:

Two arcs join correctl
The *line_width* is greater than zero
The arcs intersect

Otherwise, the intersecting pixels of intersecting arcs are drawn multiple times.

Specifying an arc with one endpoint and a clockwise extent draws the same pixels as specifying the other endpoint and an equivalent counterclockwise extent as it affects joins.

If the last point in one arc coincides with the first point in the following arc, the two arcs join correctly. If the first point in the first arc coincides with the last point in the last arc, the two arcs join correctly. By specifying one axis to be zero, a horizontal or vertical line can be drawn. Angles are computed based on the coordinate system and ignore the aspect ratio.

XDrawArcs uses the graphics context components: *function, plane_mask, line_width, line_style, cap_style, join_style, fill_style, subwindow_mode, clip_x_origin, clip_y_origin*, and *clip_mask*. This function also uses the graphics context mode-dependent components: *foreground, background, tile, stipple, ts_x_origin, ts_y_origin, dash_offset*, and *dash_list*. See "Manipulating Graphics Context or State" in topic 1.7.8.

XDrawArcs can generate the event errors **BadDrawable**, **BadGC**, and **BadMatch**.

¦ Copyright IBM Corp. 1985, 1991

2.4.123 XDrawImageString

XDrawImageString(*display, drawable, gc, x, y, string, length*)
Display **display*;
Drawable *drawable*;
GC *gc*;
int *x, y*;
char **string*;
int *length*;

display Specifies the connection to the X Server.

drawable Specifies the drawable.

gc Specifies the graphics context.

x, y Specifies the *x* and *y* coordinates, which are relative to the origin of the specified drawable. These coordinates define the baseline starting position for the image intial text character.

string Specifies the character string.

length Specifies the number of characters in the string argument.

XDrawImageString draws 8-bit image text characters in the drawable specified. It also modifies the foreground and background pixels in the characters.

XDrawImageString uses the graphics context components: *plane_mask, foreground, background, font, subwindow_mode, clip_x_origin, clip_y_origin*, and *clip_mask*. See "Manipulating Graphics Context or State" in topic 1.7.8.

This routine can generate the event errors **BadDrawable**, **BadGC**, and **BadMatch**.

¦ Copyright IBM Corp. 1985, 1991

2.4.124 XDrawImageString16

XDrawImageString16(*display, drawable, gc, x, y, string, length*)
Display **display*;
Drawable *drawable*;
GC *gc*;
int *x, y*;
XChar2b **string*;
int *length*;

display Specifies the connection to the X Server.

drawable Specifies the drawable.

gc Specifies the graphics context.

x, y Specifies the *x* and *y* coordinates, which are relative to the origin of the specified drawable. These coordinates define the baseline starting position for the image initial text character.

string Specifies the character string.

length Specifies the number of characters in the string argument.

XDrawImageString16 draws 16-bit image text characters in the drawable specified. It also modifies the foreground and background pixels in the characters. First fill a destination rectangle with the background pixel defined in the **GC** and then paint the text with the foreground pixel.

The upper-left corner of the filled rectangle is at

[x, y - font_ascent]

The width is at

overall_width.

The height is at

font_ascent + font_descent

The *overall_width*, *font_ascent* and *font_descent* are returned using *gc* and *string*. See "Manipulating Fonts" in topic 1.9.

The *string* formal parameter for **XDrawImageString16** is of type **XChar2b**. The X Server interprets each **XChar2b** as a 16-bit number that has been transmitted with the most-significant byte first. The *byte1* member of the **XChar2b** structure is taken as the most-significant byte.

XDrawImageString16 uses the graphics context components: *plane_mask, foreground, background, font, subwindow_mode, clip_x_origin, clip_y_origin*, and *clip_mask*. See "Manipulating Graphics Context or State" in topic 1.7.8.

This routine can generate the event errors **BadDrawable**, **BadGC**, and **BadMatch**.

¦ Copyright IBM Corp. 1985, 1991

2.4.125 XDrawLine

XDrawLine(*display, drawable, gc, x1, y1, x2, y2*)
Display **display*;
Drawable *drawable*;
GC *gc*;
int *x1, y1, x2, y2*;

display Specifies the connection to the X server.

drawable Specifies the drawable.

gc Specifies the graphics context.

x1, y1, x2, y2 Specifies the points used to connect the line.

XDrawLine draws a line connecting point *x1*, *y1* to point *x2*, *y2* in a specified drawable with the components of the specified graphics context. No joining is performed at coincident end points. For any given line, no pixel is drawn more than once. If lines intersect, the intersecting pixels are drawn multiple times.

XDrawLine uses the graphics context components: *function, plane_mask, line_width, line_style, cap_style, fill_style, subwindow_mode, clip_x_origin, clip_y_origin*, and *clip_mask*. It also uses the graphics context mode-dependent components: *foreground, background, tile, stipple, ts_x_origin, ts_y_origin, dash_offset*, and *dash_list*. See "Manipulating Graphics Context or State" in topic 1.7.8.

XDrawLine can generate the event errors **BadDrawable**, **BadGC**, and **BadMatch**.

¦ Copyright IBM Corp. 1985, 1991

2.4.126 XDrawLines

XDrawLines(*display, drawable, gc, points, npoints, mode*)
Display **display*;
Drawable *drawable*;
GC *gc*;
XPoint **points*;
int *npoints*;
int *mode*;

display Specifies the connection to the X Server.

drawable Specifies the drawable.

gc Specifies the graphics context.

points Specifies a pointer to an array of points.

npoints Specifies the number of points in the array.

mode Specifies the coordinate mode.

XDrawLines draws multiple lines in the specified drawable with the components of the specified graphics context. It uses these components to draw **npoints-1** lines between each pair of points **(point[i], point[i+1])** in the array of **XPoint** structures. See "Drawing Points, Lines, Rectangles, and Arcs" in topic 1.8.3 for more information on the **XPoint** data structure.

The lines are drawn in the order listed in the array. The lines join correctly at all intermediate points. If the first and the last points coincide, the first and last lines also join correctly. For any given line, no pixel is drawn more than once.

If thin (zero line width) lines intersect, the intersecting pixels will be drawn multiple times. If wide lines intersect, the intersecting pixels are drawn only once, as though the entire PolyLine were a single filled shape. The following is an example of a five-point Polyline:

```
[x,y,] [x+width,y] [x+width,y+height] [x,y+height] [x,y]
```

The *mode* argument determines if the points are relative to the origin of the drawable or to the previous point:

If *mode* is **CoordModeOrigin**, all points after the first are relative to the origin of the drawable. The first point is always relative to the origin of the drawable.

If *mode* is **CoordModePrevious**, all points after the first are relative to the previous point.

XDrawLines uses the graphics context components: *function, plane_mask, line_width, line_style, cap_style, fill_style, subwindow_mode, clip_x_origin, clip_y_origin, clip_mask* and *join_style*. It also uses the graphics context mode-dependent components: *foreground, background, tile, stipple, ts_x_origin, ts_y_origin, dash_offset*, and *dash_list*. See "Manipulating Graphics Context or State" in topic 1.7.8.

XDrawLines can generate the event errors **BadDrawable**, **BadGC**, **BadMatch**, and **BadValue**.

¦ Copyright IBM Corp. 1985, 1991

2.4.127 XDrawPoint

XDrawPoint(*display, drawable, gc, x, y*)
Display **display*;
Drawable *drawable*;
GC *gc*;
int *x*, *y*;

display Specifies the connection to the X server.

drawable Specifies the drawable.

gc Specifies the graphics context.

x, y Specifies the *x* and *y* coordinates which specify where the point will be drawn.

XDrawPoint draws a single point for a specified drawable with the foreground pixel and function components of the graphics context.

XDrawPoint uses the graphics context components: *function, plane_mask, foreground, subwindow_mode, clip_x_origin, clip_y_origin*, and *clip_mask*. See "Manipulating Graphics Context or State" in topic 1.7.8.

XDrawPoint can generate the event errors **BadDrawable**, **BadGC**, and **BadMatch**.

¦ Copyright IBM Corp. 1985, 1991

2.4.128 XDrawPoints

XDrawPoints(*display, drawable, gc, points, npoints, mode*)
Display **display*;
Drawable *drawable*;
GC *gc*;
XPoint **points*;
int *npoints*;
int *mode*;

display Specifies the connection to the X server.

drawable Specifies the drawable.

gc Specifies the graphics context.

points Specifies a pointer to an array of points.

npoints Specifies the number of points in the array.

mode Specifies the coordinate mode.

XDrawPoints draws multiple points in the specified drawable with the foreground pixel and function components from an an array of **XPoint** structures. **XDrawPoints** draws the points in the order listed in the array. See "Drawing Points, Lines, Rectangles, and Arcs" in topic 1.8.3 for information about the **XPoint** data structure.

The *mode* argument indicates whether the points are relative to the origin of the drawable or to the previous point:

If *mode* is **CoordModeOrigin**, all points after the first are relative to the origin of the drawable. The first point is always relative to the origin of the drawable.

If *mode* is **CoordModePrevious**, all points after the first point are relative to the previous point.

XDrawPoints uses the graphics context components: *function, plane_mask, foreground, subwindow_mode, clip_x_origin, clip_y_origin*, and *clip_mask*. See "Manipulating Graphics Context or State" in topic 1.7.8.

XDrawPoints can generate the event errors **BadDrawable**, **BadGC**, **BadMatch**, and **BadValue**.

¦ Copyright IBM Corp. 1985, 1991

2.4.129 XDrawRectangle

XDrawRectangle(*display, drawable, gc, x, y, width, height*)
Display **display*;
Drawable *drawable*;
GC *gc*;
int *x*, *y*;
unsigned int *width*, *height*;

display	Specifies the connection to the X Server.
drawable	Specifies the drawable.
gc	Specifies the graphics context.
x, y	Specifies the *x* and *y* coordinates. These coordinates, which are relative to the origin of the drawable, define the upper-left corner of the rectangle.
width, height	Specifies the *width* and *height* which define the outline of the rectangle.

XDrawRectangle draws the outline of a single rectangle as if a five-point PolyLine were specified for the rectangle. The following is an example of a five-point Polyline:

```
  [x,y,] [x+width,y] [x+width,y+height] [x,y+height] [x,y]
```

For the specified rectangle, no pixel is drawn more than once.

XDrawRectangle uses the graphics context components: *function, plane_mask, line_width, line_style, join_style, fill_style, subwindow_mode, clip_x_origin, clip_y_origin*, and *clip_mask*. It also uses the graphics context mode-dependent components: *foreground, background, tile, stipple, ts_x_origin, ts_y_origin, dash_offset*, and *dash_list*. See "Manipulating Graphics Context or State" in topic 1.7.8.

XDrawRectangle can generate the event errors **BadDrawable**, **BadGC**, and **BadMatch**.

¦ Copyright IBM Corp. 1985, 1991

2.4.130 XDrawRectangles

XDrawRectangles(*display, drawable, gc, rectangles, nrectangles*)
Display **display*;
Drawable *drawable*;
GC *gc*;
XRectangle *rectangles*[];
int *nrectangles*;

display Specifies the connection to the X Server.

drawable Specifies the drawable.

gc Specifies the graphics context.

rectangles Specifies a pointer to an array of rectangles.

nrectangles Specifies the number of rectangles in the array.

XDrawRectangles draws the outline of multiple rectangles in the specified drawable as if a five-point PolyLine were specified for each rectangle. It draws the rectangles in the order listed in the array. The following is an example of a five-point Polyline:

```
[x,y,] [x+width,y] [x+width,y+height] [x,y+height] [x,y]
```

For the specified rectangles, no pixel is drawn more than once. If the rectangles intersect, the intersecting pixels are drawn multiple times. (See "Drawing Points, Lines, Rectangles, and Arcs" in topic 1.8.3 for information about the **XRectangle** data structure.)

XDrawRectangles uses the graphics context components: *function, plane_mask, line_width, line_style, join_style, fill_style, subwindow_mode, clip_x_origin, clip_y_origin*, and *clip_mask*. It also uses the graphics context mode-dependent components: *foreground, background, tile, stipple, ts_x_origin, ts_y_origin, dash_offset*, and *dash_list*. See "Manipulating Graphics Context or State" in topic 1.7.8.

XDrawRectangles can generate the event errors **BadDrawable**, **BadGC**, and **BadMatch**.

¦ Copyright IBM Corp. 1985, 1991

2.4.131 XDrawSegments

XDrawSegments(*display, drawable, gc, segments, nsegments*)
Display **display*;
Drawable *drawable*;
GC *gc*;
XSegment **segments*;
int *nsegments*;

display Specifies the connection to the X Server.

drawable Specifies the drawable.

gc Specifies the graphics context.

segments Specifies a pointer to an array of segments.

nsegments Specifies the number of segments in the array.

XDrawSegments draws multiple, but not necessarily connected, lines in the specified drawable. It draws a line for each segment between *x1, y1* and *x2, y2* as defined in the **XSegment** data structure. It draws the lines in the order listed in the array of **XSegment** structures. No joining is performed at coincident end points. For any given line, no pixel is drawn more than once. If lines intersect, the intersecting pixels are drawn multiple times. (See "Drawing Points, Lines, Rectangles, and Arcs" in topic 1.8.3 for information about the **XSegment** structures.)

XDrawSegments uses the graphics context components: *function, plane_mask, line_width, line_style, cap_style, fill_style, subwindow_mode, clip_x_origin, clip_y_origin*, and *clip_mask*. It also uses the graphics context mode-dependent components: *foreground, background, tile, stipple, ts_x_origin, ts_y_origin, dash_offset*, and *dash_list*. See "Manipulating Graphics Context or State" in topic 1.7.8.

XDrawSegments can generate the event errors **BadDrawable**, **BadGC**, and **BadMatch**.

¦ Copyright IBM Corp. 1985, 1991

2.4.132 XDrawString

XDrawString(*display, drawable, gc, x, y, string, length*)
Display **display*;
Drawable *drawable*;
GC *gc*;
int *x, y*;
char **string*;
int *length*;

display Specifies the connection to the X Server.

drawable Specifies the drawable.

gc Specifies the graphics context.

x, y Specifies the *x* and *y* coordinates, which are relative to the origin of the specified drawable. These coordinates define the baseline starting position for the initial character.

string Specifies the character string.

length Specifies the number of characters in the string argument.

XDrawString draws 8-bit text characters in the specified drawable. Each character image, as defined by the font in the **GC**, is treated as an additional mask for a fill operation on the drawable. See "Manipulating Fonts" in topic 1.9.

This function uses the graphics context components: *function, plane_mask, fill_style, font, subwindow_mode, clip_x_origin, clip_y_origin*, and *clip_mask*. It also uses the graphics context mode-dependent components: *foreground, background, tile, stipple, ts_x_origin*, and *ts_y_origin*. See "Manipulating Graphics Context or State" in topic 1.7.8.

XDrawString can generate the event errors **BadDrawable**, **BadFont**, **BadGC**, and **BadMatch**.

¦ Copyright IBM Corp. 1985, 1991

2.4.133 XDrawString16

XDrawString16(*display, drawable, gc, x, y, string, length*)
Display **display*;
Drawable *drawable*;
GC *gc*;
int *x, y*;
XChar2b **string*;
int *length*;

display Specifies the connection to the X Server.

drawable Specifies the drawable.

gc Specifies the graphics context.

x, y Specifies the *x* and *y* coordinates, which are relative to the origin of the specified drawable. These coordinates define the baseline starting position for the initial character.

string Specifies the character string.

length Specifies the number of characters in the string argument.

XDrawString16 draws 16-bit characters in the specified drawable. Each character image, as defined by the font in the **GC**, is treated as an additional mask for a fill operation on the drawable. See "Manipulating Fonts" in topic 1.9.

This function uses the graphics context components: *function, plane_mask, fill_style, font, subwindow_mode, clip_x_origin, clip_y_origin*, and *clip_mask*. It also uses the graphics context mode-dependent components: *foreground, background, tile, stipple, ts_x_origin*, and *ts_y_origin*. See "Manipulating Graphics Context or State" in topic 1.7.8.

XDrawString16 can generate the event errors **BadDrawable**, **BadFont**, **BadGC**, and **BadMatch**.

¦ Copyright IBM Corp. 1985, 1991

2.4.134 XDrawText

XDrawText(*display, drawable, gc, x, y, items, nitems*)
Display **display*;
Drawable *drawable*;
GC *gc*;
int *x, y*;
XTextItem **items*;
int *nitems*;

display Specifies the connection to the X Server.

drawable Specifies the drawable.

gc Specifies the graphics context.

x, y Specifies the *x* and *y* coordinates. These coordinates, which are relative to the origin of the specified drawable, define the baseline starting position for the initial character.

items Specifies a pointer to an array of text items.

nitems Specifies the number of text items in the array.

XDrawText draws 8-bit polytext characters for a specified drawable. Each text item is processed in turn. This function uses the data structure **XTextItem**. See "Drawing Text Characters" in topic 1.9.2.

If the *font* member is **None**, the font is changed before printing and is stored in the **GC**. During text drawing, if an error is generated, the font in the **GC** is undefined.
If the *font* member is not **None**, the font is stored in the **GC** and used for subsequent text.
A text element delta specifies an additional change in the positio along the *x* axis before the string is drawn. The delta is always added to the character origin and is not dependent on any characteristics of the font.
Each character image, as defined by the font in the **GC**, 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 se to one. If the previous text items have been drawn, it can generate an error.

XDrawText uses the graphics context components: *function, plane_mask, fill_style, font, subwindow_mode, clip_x_origin, clip_y_origin* and *clip_mask*. It also uses the graphics context mode-dependent components: *foreground, background, tile stipple, ts_x_origin*, and *ts_y_origin*. See "Manipulating Graphics Context or State" in topic 1.7.8.

XDrawText can generate the event errors **BadDrawable**, **BadFont**, **BadGC**, and **BadMatch**.

¦ Copyright IBM Corp. 1985, 1991

2.4.135 XDrawText16

XDrawText16(*display, drawable, gc, x, y, items, nitems*)
Display **display*;
Drawable *drawable*;
GC *gc*;
int *x, y*;
XTextItem16 **items*;
int *nitems*;

display Specifies the connection to the X Server.

drawable Specifies the drawable.

gc Specifies the graphics context.

x, y Specifies the *x* and *y* coordinates. These coordinates, which are relative to the origin of the specified drawable, define the baseline starting position for the initial character.

items Specifies a pointer to an array of text items.

nitems Specifies the number of text items in the array.

XDrawText16 draws 16-bit polytext characters in the specified drawable. Each text item is processed in turn. This function uses the data structure **XTextItem16**. See "Drawing Text Characters" in topic 1.9.2.

If the *font* member is **None**, the font is changed before printing and is stored in the **GC**. During text drawing, if an error is generated, the font in the **GC** is undefined.
If the *font* member is not **None**, the font is stored in the **GC** and used for subsequent text.
A text element delta specifies an additional change in the positio along the *x* axis before the string is drawn. The delta is always added to the character origin and is not dependent on any characteristics of the font.
Each character image, as defined by the font in the **GC**, 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 se to one. If the previous text items have been drawn, it can generate an error.

XDrawText16 uses the graphics context components: *function, plane_mask, fill_style, font, subwindow_mode, clip_x_origin, clip_y_origin*, and *clip_mask*. It also uses the graphics context mode-dependent components: *foreground, background, tile, stipple, ts_x_origin*, and *ts_y_origin*. See "Manipulating Graphics Context or State" in topic 1.7.8.

XDrawText16 can generate the event errors **BadDrawable**, **BadFont**, **BadGC**, and **BadMatch**.

¦ Copyright IBM Corp. 1985, 1991

2.4.136 XEmptyRegion

int XEmptyRegion(*region*)
Region *region*;

region Specifies the region.

XEmptyRegion determines if a specified region is empty. If the region is empty, it returns a non-zero.

¦ Copyright IBM Corp. 1985, 1991

2.4.137 XEnableAccessControl

XEnableAccessControl(*display*)
Display **display*;

display Specifies the connection to the X Server.

XEnableAccessControl enables the use of the access control list at connection setups. For this function to execute successfully, the client application must reside on the same host as the X Server and have the necessary permission in the initial authorization at connection setup. See "Controlling Host Access" in topic 1.10.6.

This routine can generate the event error **BadAccess**.

¦ Copyright IBM Corp. 1985, 1991

2.4.138 XEqualRegion

int XEqualRegion(*region1, region2*)
Region *region1, region2*;

region1, region2 Specifies the two regions to use to compare offset, size, and shape.

XEqualRegion determines if two regions have the same offset, size, and shape. If the two regions are identical, it returns a non-zero.

¦ Copyright IBM Corp. 1985, 1991

2.4.139 XEventsQueued

int XEventsQueued(*display, mode*)
Display **display*;
int *mode*;

display Specifies the connection to the X Server.

mode Specifies the mode.

XEventsQueued checks the number of events in the event queue. If there are events in the queue already, it returns immediately without I/O.

The *mode* variable can be **QueuedAlready**, **QueuedAfterFlush**, **QueuedAfterReading**.

If *mode* is set to **QueuedAlready**, **XEventsQueued** returns the number of events already in the event queue and does not perform a system call. In this *mode*, **XEventsQueued** is identical to **XQLength**.

If *mode* is **QueuedAfterFlush** and the number of events is non-zero, **XEventsQueued** returns the number of events already in the queue. If there are no events in the queue, it:

- Flushes the output buffer
- Attempts to read more events out of the application's connection
- Returns the number of events read.

XEventsQueued is identical to **XPending** when *mode* is **QueuedAfterFlush**.

If *mode* is **QueuedAfterReading** and the number of events is non-zero, **XEventsQueued** returns the number of events already in the queue. If there are no events in the queue, it:

- Attempts to read more events out of the application's connection without flushing the output buffer
- Returns the number of events read.

¦ Copyright IBM Corp. 1985, 1991

2.4.140 XFetchBuffer

char *XFetchBuffer(*display, nbytes_return , return_buffer*)
Display **display*;
int **nbytes_return*;
int *return_buffer*;

display Specifies the connection to the X Server.

nbytes_return Returns the number of bytes in the string in the buffer.

return_buffer Specifies the buffer from which the stored data will be returned.

XFetchBuffer returns data from a specified cut buffer. If there is no data in the buffer, it returns zero to the *nbytes* argument.

This routine can generate the event error **BadValue**.

¦ Copyright IBM Corp. 1985, 1991

2.4.141 XFetchBytes

char *XFetchBytes(*display, nbytes_return*)
Display **display*;
int **nbytes_return*;

display Specifies the connection to the X Server.

nbytes_return Returns the number of bytes in the string in the buffer.

XFetchBytes returns data from cut buffer zero. It returns the number of bytes in the *nbytes_return* argument. If there is no data in the buffer, it returns **NULL** and sets *nbytes* to zero. The appropriate amount of storage is allocated and the pointer is returned. The client must free this storage when it is finished by calling **XFree**.

The cut buffer does not necessarily contain text. It may contain embedded null bytes and may not terminate with a null byte.

XFetchBytes can generate the event error **BadWindow**.

¦ Copyright IBM Corp. 1985, 1991

2.4.142 XFetchName

int XFetchName(*display, window, window_name_return*)
Display **display*;
Window *window*;
char ***window_name_return*;

display	Specifies the connection to the X Server.
window	Specifies the window ID for the window where the pointer will be returned.
window_name_return	Returns a pointer (a null-terminated string) to the window specified.

XFetchName obtains a window name. (A client should free the name string when it is finished with this function.)

If **XFetchName** succeeds, it returns a non-zero.

If **XFetchName** fails, it returns a zero.

XFetchName can fail if no name was set for the specified window.

If the **WM_NAME** property has not been set for this window, **XFetchName** sets it to **NULL**. See "Using Predefined Property Functions" in topic 1.12.

This routine can generate the event error **BadWindow**.

¦ Copyright IBM Corp. 1985, 1991

2.4.143 XFillArc

XFillArc(*display, drawable, gc, x, y, width, height, angle1, angle2*)
Display **display*;
Drawable *drawable*;
GC *gc*;
int *x*, *y*;
unsigned int *width*, *height*;
int *angle1*, *angle2*;

display	Specifies the connection to the X Server.
drawable	Specifies the drawable.
gc	Specifies the graphics context.
x, y	Specifies the *x* and *y* coordinates. These coordinates, which are relative to the origin of the specified drawable, define the upper-left corner of the rectangle.
width, height	Specifies the *width* and *height* which define the major and minor axes of the arc.
angle1	Specifies the start of the arc relative to the 3 o'clock position from the center, in units of degrees multiplied by 64.
angle2	Specifies the path and extent of the arc relative to the start of the arc, in units of degrees multiplied by 64.

XFillArc fills a single arc in the specified drawable. It fills the region closed by the infinitely thin path described by the specified arc and, depending on the *arc_mode* specified in the graphics context, one or two line segments. The single line segment joining the endpoints of the arc is used for **ArcChord**. The two line segments joining the endpoints of the arc with the center point are used for **ArcPieSlice**.

XFillArc uses the graphics context components: *function, plane_mask, fill_style, arc_mode, subwindow_mode, clip_x_origin, clip_y_origin*, and *clip_mask*. It also uses the graphics context mode-dependent components: *foreground, background, tile, stipple, ts_x_origin*, and *ts_y_origin*. See "Manipulating Graphics Context or State" in topic 1.7.8.

XFillArc can generate the event errors **BadDrawable**, **BadGC**, and **BadMatch**.

¦ Copyright IBM Corp. 1985, 1991

2.4.144 XFillArcs

XFillArcs(*display, drawable, gc, arcs, narcs*)
Display **display*;
Drawable *drawable*;
GC *gc*;
XArc **arcs*;
int *narcs*;

display Specifies the connection to the X Server.

drawable Specifies the drawable.

gc Specifies the graphics context.

arcs Specifies a pointer to an array of arcs.

narcs Specifies the number of arcs in the array.

XFillArcs fills multiple arcs in the specified drawable from an array of **XArc** structures. It fills the arcs in the order listed in the array. For each arc, it fills the region closed by the infinitely thin path described by the specified arc and, depending on the *arc_mode* specified in the graphics context, one or two line segments. The single line segment joining the endpoints of the arc is used for **ArcChord**. The two line segments joining the endpoints of the arc with the center point are used for **ArcPieSlice**.

For any given arc, no pixel is drawn more than once. If regions intersect, the intersecting pixels are drawn multiple times. See "Drawing Points, Lines, Rectangles, and Arcs" in topic 1.8.3 for information about **XArc**.

XFillArcs uses the graphics context components: *function, plane_mask, fill_style, arc_mode, subwindow_mode, clip_x_origin, clip_y_origin*, and *clip_mask*. It also uses the graphics context mode-dependent components: *foreground, background, tile, stipple, ts_x_origin*, and *ts_y_origin*. See "Manipulating Graphics Context or State" in topic 1.7.8.

XFillArcs can generate the event errors **BadDrawable**, **BadGC**, and **BadMatch**.

¦ Copyright IBM Corp. 1985, 1991

2.4.145 XFillPolygon

XFillPolygon(*display, drawable, gc, points, npoints, shape, mode*)
Display **display*;
Drawable *drawable*;
GC *gc*;
XPoint **points*;
int *npoints*;
int *shape*;
int *mode*;

display Specifies the connection to the X Server.

drawable Specifies the drawable.

gc Specifies the graphics context.

points Specifies a pointer to an array of points. See "Drawing Points, Lines, Rectangles, and Arcs" in topic 1.8.3.

npoints Specifies the number of points in the array.

shape Specifies the shape.

mode Specifies the coordinate mode.

XFillPolygon fills a polygon area in the specified drawable. It fills the region closed by the specified path. The path is closed automatically if the last point in the list does not coincide with the first point. No pixel of the region is drawn more than once.

The *shape* argument can be one of the following:

If *shape* is **Complex**, the path may self-intersect.

If *shape* is **Nonconvex**, the path does not self-intersect, but it is not totally convex. If **Nonconvex** is specified for a self-intersecting path, the graphics results are undefined.

If *shape* is **Convex**, the path is totally convex. If **Convex** is specified for a path that is not convex, the graphics results are undefined. The *fill_rule* member of the **GC** controls the filling behavior of self-intersecting polygons.

The *mode* argument determines if the points are relative to the origin of the drawable or to the previous point:

If *mode* is **CoordModeOrigin**, all points after the first are relative to the origin of the drawable. The first point is always relative to the origin of the drawable.

If *mode* is **CoordModePrevious**, all points after the first are relative to the previous point.

XFillPolygon uses the graphics context components: *function, plane_mask, fill_style, fill_rule, subwindow_mode, clip_x_origin, clip_y_origin*, and *clip_mask*. It also uses the graphics context mode_dependent components: *foreground, tile, stipple, ts_x_origin*, and *ts_y_origin*. See "Manipulating Graphics Context or State" in topic 1.7.8.

¦ Copyright IBM Corp. 1985, 1991

XFillPolygon can generate the event errors **BadDrawable**, **BadGC**, **BadMatch**, and **BadValue**.

¦ Copyright IBM Corp. 1985, 1991

2.4.146 XFillRectangle

XFillRectangle(*display, drawable, gc, x, y, width, height*)
Display **display*;
Drawable *drawable*;
GC *gc*;
int *x*, *y*;
unsigned int *width*, *height*;

display	Specifies the connection to the X Server.
drawable	Specifies the drawable.
gc	Specifies the graphics context.
x, y	Specifies the *x* and *y* coordinates. These coordinates, which are relative to the origin of the specified drawable, define the upper-left corner of the rectangle.
width, height	Specifies the *width* and *height* which define the dimensions of the rectangle.

XFillRectangle fills a single rectangular area as specified. This function uses the graphics context components: *function, plane_mask, fill_style, fill_rule, subwindow_mode, clip_x_origin, clip_y_origin*, and *clip_mask*. It also uses the graphics context mode_dependent components: *foreground, background, tile, stipple, ts_x_origin*, and *ts_y_origin*. See "Manipulating Graphics Context or State" in topic 1.7.8.

XFillRectangle can generate the event errors **BadDrawable**, **BadGC**, and **BadMatch**.

¦ Copyright IBM Corp. 1985, 1991

2.4.147 XFillRectangles

XFillRectangles(*display, drawable, gc, rectangles, nrectangles*)
Display **display*;
Drawable *drawable*;
GC *gc*;
XRectangle **rectangles*;
int *nrectangles*;

display Specifies the connection to the X Server.

drawable Specifies the drawable.

gc Specifies the graphics context.

rectangles Specifies a pointer to an array of rectangles.

nrectangles Specifies the number of rectangles in the array.

XFillRectangles fills multiple rectangular areas in the specified drawable as if a four-point XFillPolygon was specified for each rectangle. An example of a four-point XFillPolygon is:

```
[x,y] [x+width,y] [x+width,y+height] [x,y+height]
```

For **XFillRectangles**, specify an array of **XRectangle** structures. (See "Drawing Points, Lines, Rectangles, and Arcs" in topic 1.8.3.) This function fills the rectangles in the order listed in the array. For any given rectangle, no pixel is drawn more than once. If rectangles intersect, the intersecting pixels are drawn multiple times.

XFillRectangles uses the graphics context components: *function, plane_mask, fill_style, fill_rule, subwindow_mode, clip_x_origin, clip_y_origin*, and *clip_mask*. It also uses the graphics context mode_dependent components: *foreground, background, tile, stipple, ts_x_origin*, and *ts_y_origin*. See "Manipulating Graphics Context or State" in topic 1.7.8.

XFillRectangle can generate the event errors **BadDrawable**, **BadGC**, and **BadMatch**.

¦ Copyright IBM Corp. 1985, 1991

2.4.148 XFindContext

int XFindContext(*display, window, context, data_return*)
Display **display*;
Window *window*;
XContext *context*;
caddr_t **data_return*;

display Specifies the connection to the X Server.

window Specifies the window ID for the window with which the data is associated.

context Specifies the context type to which the data belongs.

data_return Returns a pointer to the data.

XFindContext gets the data associated with a window and context type. If it is successful, it returns a zero. If it is not successful, it returns a non-zero.

XFindContext can generate the error **XCNOENT** which indicates that the context was not found.

¦ Copyright IBM Corp. 1985, 1991

2.4.149 XFlush

XFlush(*display*)
Display **display*;

display Specifies the connection to the X Server.

XFlush flushes the output buffer. Most client applications do not use this function because the output buffer is automatically flushed the next time an event or reply is read from the output buffer by a call to **XPending**, **XNextEvent**, or **XWindowEvent**.

¦ Copyright IBM Corp. 1985, 1991

2.4.150 XForceScreenSaver

XForceScreenSaver(*display, mode*)
Display **display*;
int *mode*;

display Specifies the connection to the X Server.

mode Specifies the screen saver mode.

XForceScreenSaver forces the screen saver on or off by applying a mode to the screen saver. The *mode* argument can be **ScreenSaverActive** or **ScreenSaverReset**.

If *mode* is **ScreenSaverActive** and the screen saver is currently deactivated, the screen saver is activated, even if the screen saver had been disabled with a timeout of zero.

If *mode* is **ScreenSaverReset** and the screen saver currently is enabled, the screen saver is deactivated, if it was activated, and the activation timer is reset to its initial state--as if device input had been received.

For more information, see **XSetScreenSaver** in this chapter.

XForceScreenSaver can generate the event error **BadValue**.

¦ Copyright IBM Corp. 1985, 1991

2.4.151 XFree

XFree(*data*)
char **data*;

data Specifies a pointer to the data that is to be freed.

XFree frees any in-memory data that was created by an **Xlib** function.

¦ Copyright IBM Corp. 1985, 1991

2.4.152 XFreeColormap

XFreeColormap(*display, cmap*)
Display **display*;
Colormap *cmap*;

display Specifies the connection to the X Server.

cmap Specifies the colormap ID of the colormap associated with the resource ID to be deleted.

XFreeColormap deletes the association between the colormap resource ID and the colormap. However, it has no effect on a default colormap for a screen.

If *cmap* is an installed colormap for a screen, the colormap becomes uninstalled with **XFreeColormap**. See **XUninstallColormap** in this chapter.

If *cmap* is defined as the colormap for a window, **XFreeColormap** changes the colormap associated with the window to **None** and generates a **ColormapNotify** event. See "Processing Colormap State Notification Events" in topic 1.11.17.

The colors displayed for a window with a colormap of **None** are not defined.

XFreeColormap can generate the event error **BadColor**.

¦ Copyright IBM Corp. 1985, 1991

2.4.153 XFreeColors

XFreeColors(*display, cmap, pixels, npixels, planes*)
Display **display*;
Colormap *cmap*;
unsigned long *pixels[]*;
int *npixels*;
unsigned long *planes*;

display Specifies the connection to the X Server.

cmap Specifies the colormap ID.

pixels Specifies an array of pixel values which map to the cells in the specified colormap.

npixels Specifies the number of pixels.

planes Specifies the planes to be freed.

XFreeColors frees colormap cells. It frees the cells represented by pixels whose values are in the pixels array. The request frees the pixels that were allocated by the client using **XAllocColor**, **XAllocNamedColor**, **XAllocColorCells**, and **XAllocColorPlanes**. (Freeing a pixel obtained from **XAllocColorPlanes** may not allow the pixel to be reused until all the related pixels are also freed.)

The *planes* argument should not have bits in common with the pixels. The set of all pixels is produced by combining the subsets of the *planes* argument with the pixels.

The pixels specified that are allocated by the client in *cmap* are freed, even if an error is generated.

XFreeColors can generate the event errors **BadAccess**, **BadColor**, and **BadValue**.

¦ Copyright IBM Corp. 1985, 1991

2.4.154 XFreeCursor

XFreeCursor(*display, cursor*)
Display **display*;
Cursor *cursor*;

display Specifies the connection to the X Server.

cursor Specifies the cursor.

XFreeCursor deletes the association between the cursor resource ID and the specified cursor. The cursor storage is freed when no other resource references it. The specified cursor should not be referenced again.

This routine can generate the error **BadCursor**.

¦ Copyright IBM Corp. 1985, 1991

2.4.155 XFreeFont

XFreeFont(*display, font_struct*)
Display **display*;
XFontStruct **font_struct*;

display Specifies the connection to the X Server.

font_struct Specifies the storage associated with the font.

XFreeFont deletes the association between the font resource ID and the specified font. The font is freed when no other resources reference it. See "Manipulating Fonts" in topic 1.9 for information about fonts.

This routine can generate the event error **BadFont**.

¦ Copyright IBM Corp. 1985, 1991

2.4.156 XFreeFontInfo

XFreeFontInfo(*names, free_info, actual_count*)
char ***names*;
XFontStruct **free_info*;
int *actual_count*;

names Specifies a pointer to the list of font names that were returned by **XListFontsWithInfo**.

free_info Specifies a pointer to the font information that was returned by **XListFontsWithInfo**.

actual_count Specifies the actual number of matched font names that were returned by **XListFontsWithInfo**.

XFreeFontInfo frees the font information array. See "Manipulating Fonts" in topic 1.9 for information about fonts.

¦ Copyright IBM Corp. 1985, 1991

2.4.157 XFreeFontNames

XFreeFontNames(*list*)
char **list[];*

list Specifies the array of strings to be freed.

XFreeFontNames frees the array and strings returned by **XListFonts**. See "Manipulating Fonts" in topic 1.9 for information about fonts.

¦ Copyright IBM Corp. 1985, 1991

2.4.158 XFreeFontPath

XFreeFontPath(*list*)
char ***list*;

list Specifies the array of strings to be freed.

XFreeFontPath frees the data used by the array when presented the data from **XGetFontPath**. See "Manipulating Fonts" in topic 1.9 for information about fonts.

¦ Copyright IBM Corp. 1985, 1991

2.4.159 XFreeGC

XFreeGC(*display*, *gc*)
Display **display*;
GC *gc*;

display Specifies the connection to the X Server.

gc Specifies the graphics context.

XFreeGC frees and destroys the specified graphics context.

This routine can generate the event error **BadGC**.

¦ Copyright IBM Corp. 1985, 1991

2.4.160 XFreeModifiermap

XFreeModifiermap(*modmap*)
XModifierKeymap **modmap*;

modmap Specifies a pointer to the **XModifierKeymap** structure.

XFreeModifiermap frees the specified **XModifierKeymap** structure. See "Manipulating Keyboard Encoding" in topic 1.10.5.

¦ Copyright IBM Corp. 1985, 1991

2.4.161 XFreePixmap

XFreePixmap(*display*, *pixmap*)
Display **display*;
Pixmap *pixmap*;

display Specifies the connection to the X Server.

pixmap Specifies the pixmap.

XFreePixmap frees all storage associated with a specified pixmap. First, it deletes the association between the pixmap ID and the pixmap. Then, the X Server frees the pixmap storage when it is no longer referenced by other resources. The pixmap should not be referenced after this function completes.

XFreePixmap can generate the error **BadPixmap**.

¦ Copyright IBM Corp. 1985, 1991

2.4.162 XGContextFromGC

GContext XGContextFromGC(*gc*)
GC *gc*;

gc Specifies the graphics context.

XGContextFromGC obtains the **GContext** for the specified *gc*.

¦ Copyright IBM Corp. 1985, 1991

2.4.163 XGeometry

int XGeometry(*display, screen_number, position, default, bwidth, fwidth, fhei*
xadder, yadder, x_return, y_return, width_return, height_r
Display **display*;
int *screen_number*;
char **position, *default*;
unsigned int *bwidth*;
unsigned int *fwidth, fheight*;
int *xadder, yadder*;
int **x_return, *y_return*;
int **width_return, *height_return*;

display	Specifies the connection to the X Server.
screen_number	Specifies the screen number of the display.
position	Specifies the geometry specifications.
default	Specifies the geometry specifications.
bwidth	Specifies the border width.
fheight, fwidth	Specifies the font height and width in pixels (increment size).
xadder, yadder	Specifies additional interior padding needed in the window.
x_return, y_return	Returns the xoffset and yoffset determined.
width_return, height_return	Returns the width and height determined.

XGeometry parses window geometry given a position and a default position. **XGeometry** determines the placement of a window using the current format to position windows. Given a fully qualified default geometry specification and, possibly, an incompletely specified geometry specification, it will return a bitmask value as defined in the call **XParseGeometry**.

The *bwidth*, *fwidth*, *fheight*, *xadder* and *yadder* are used to compute the resulting size.

¦ Copyright IBM Corp. 1985, 1991

2.4.164 XGetAtomName

char *XGetAtomName(*display, atom*)
Display **display*;
Atom *atom;*

display Specifies the connection to the X Server.

atom Specifies the atom associated with the string name to be returned.

XGetAtomName returns a name for the specified atom identifier.

This routine can generate the event error **BadAtom**.

¦ Copyright IBM Corp. 1985, 1991

2.4.165 XGetClassHint

Status XGetClassHint(*display, window, class_hints_return*)
Display **display*;
Window *window*;
XClassHint **class_hints_return*;

display Specifies the connection to the X Server.

window Specifies the window ID.

class_hints_return Returns the **XClassHint** structure.

XGetClassHint obtains the class of the specified window.

This routine can generate the event error **BadWindow**.

¦ Copyright IBM Corp. 1985, 1991

2.4.166 XGetDefault

char *XGetDefault(*display, program, option*)
Display **display*;
char **program*;
char **option*;

display Specifies the connection to the X Server.

program Specifies the *program* name for the **Xlib** defaults which is usually argv[0].

option Specifies the option name.

XGetDefault helps the client determine the fonts, colors, and other environment defaults favored by a particular user. The strings returned by **XGetDefault** are owned by **Xlib** and should not be modified or freed by the client.

Defaults are usually loaded into the **RESOURCE_MANAGER** property on the root window at login. If no such property exists, a resource file in the user's home directory is loaded. This file is **$HOME/.Xdefaults**.

After loading these defaults, **XGetDefault** merges additional defaults specified by **XENVIRONMENT** environment variable. If **XENVIRONMENT** is defined, it contains a full path name for the additional resource file. If **XENVIRONMENT** is not defined, **XGetDefault** looks for **$HOME/.Xdefaults**-*name*, where *name* specifies the name of the system running the application.

XGetDefault returns the value **NULL** if the option name specified in this argument does not exist.

¦ Copyright IBM Corp. 1985, 1991

2.4.167 XGetErrorDatabaseText

XGetErrorDatabaseText(*display, name, message, default_string, buffer_return, length*)
Display *display*;
char **name*, **message*;
char **default_string*;
char **buffer_return*;
int *length*;

display Specifies the connection to the X Server.

name Specifies the application name.

message Specifies the type of the error message to be returned.

default_string Specifies the default error message.

buffer_return Returns the error description.

length Specifies the size of *buffer_return*.

XGetErrorDatabaseText returns a message or the default message from the error message database. It uses a pair of strings as keys and searches the resource manager for the string. Then:

If a string is found, it returns it in the *buffer_return* argument.

If no string is found in the error database, it returns th *default_string* to the *buffer_return* argument.

Three predefined sets of names are used by **Xlib** to report errors:

XProtoError The protocol error number used as a string for the *message* argument.

Xlib message The message strings used internally by the library.

XRequestMajor The major request protocol number used for the *message* argument.

The error message database is **/usr/lpp/X11/messages/XErrorDB** on an AIX-based system.

¦ Copyright IBM Corp. 1985, 1991

2.4.168 XGetErrorText

XGetErrorText(*display, code, buffer_return, length*)
Display **display*;
int *code*;
char **buffer_return*;
int *length*;

display Specifies the connection to the X Server.

code Specifies the error code for which you want to obtain a description.

buffer_return Returns the error description.

length Specifies the size of *buffer_return*.

XGetErrorText copies a null-terminated string describing the specified error code into the specified buffer. It is recommended you use this routine to obtain an error description because extensions to the **Xlib** library may define their own error codes and error strings.

¦ Copyright IBM Corp. 1985, 1991

2.4.169 XGetFontPath

char ****XGetFontPath**(*display, npaths_return*)
Display **display*;
int **npaths_return*;

display Specifies the connection to the X Server.

npaths_return Returns the number of strings in the font path array.

XGetFontPath allocates and returns an array of strings containing the search path. The data in the font path should be freed when it is no longer needed.

¦ Copyright IBM Corp. 1985, 1991

2.4.170 XGetFontProperty

Bool XGetFontProperty(*font_struct, atom, value_return*)
XFontStruct **font_struct*;
Atom *atom;*
unsigned long **value_return*;

font_struct Specifies the storage associated with the font.

atom Specifies the atom associated with the string name to be returned.

value_return Returns the value of the font property.

XGetFontProperty returns the value of the specified font property. If the atom was not defined, it returns a zero. If the atom was defined, it returns a one. There is a set of predefined atoms for font properties in **<X11/Xatom.h>**. This set contains the standard properties associated with a font.

For more information on fonts, see "Manipulating Fonts" in topic 1.9.

¦ Copyright IBM Corp. 1985, 1991

2.4.171 XGetGeometry

Status XGetGeometry(*display, drawable, root_return, x_return, y_return, width_*
height_return, border_width_return, depth_return
Display **display*;
Drawable *drawable*;
Window **root_return*;
int **x_return, *y_return*;
unsigned int **width_return, *height_return*;
unsigned int **border_width_return*;
unsigned int **depth_return*;

display	Specifies the connection to the X Server.
drawable	Specifies the drawable, a window or a pixmap.
root_return	Returns the root window ID for the specified window.
x_return, y_return	Returns the *x* and *y* coordinates which define the location of the drawable.
width_return, height_return	Returns the *width* and *height* of the drawable.
border_width_return	Returns the border width.
depth_return	Returns the depth of the pixmap, bits per pixel for the drawable.

XGetGeometry obtains the current geometry of the specified drawable. It returns the root ID and the current geometry. This routine can be used with **InputOnly** windows.

The *x_return* and *y_return* specify the upper-left outer corner relative to the parent origin if the drawable is a window. For pixmaps, these coordinates are always zero.

The *width_return* and the *height_return* specify the inside of the window, excluding the border.

The *border_width_return* returns the width (in pixels) of the border if the drawable is a window. It returns a zero if the drawable is a pixmap.

XGetGeometry can generate the event error **BadDrawable**.

¦ Copyright IBM Corp. 1985, 1991

2.4.172 XGetIconName

int XGetIconName(*display, window, icon_name_return*)
Display **display*;
Window *window*;
char ***icon_name_return*;

display	Specifies the connection to the X Server.
window	Specifies the window ID for the target window.
icon_name_return	Returns a pointer to the name (a null-terminated string) to be displayed in the icon window.

XGetIconName obtains the name for the icon window. (A client must free the icon name when this function is completed.)

If **XGetIconName** succeeds, it returns a non-zero.

If the icon window does not have a name, **XGetIconName** fails and returns a zero.

If the icon window was not assigned a name, **XGetIconName** sets the *icon_name_return* argument to **NULL**.

XGetIconName can generate the event error **BadWindow**.

¦ Copyright IBM Corp. 1985, 1991

2.4.173 XGetIconSizes

Status XGetIconSizes(*display, window, size_list_return, count_return*)
Display **display*;
Window *window*;
XIconSize ***size_list_return*;
int **count_return*;

display	Specifies the connection to the X Server.
window	Specifies the window ID.
size_list_return	Returns a pointer to the size list.
count_return	Returns the number of items in the size list.

XGetIconSizes returns the value of the icon sizes atom. This function should be called by a client to determine the best icon sizes for the window manager. Then, the client should use **XSetWMHints** to supply the window manager with an icon pixmap or window in one of the supported sizes. (See "Setting and Getting Window Manager Sizing Hints" in topic 1.12.5 for information.)

If a window manager has not set icon sizes, **XGetIconSizes** returns a zero. Otherwise, it returns a non-zero.

XGetIconSizes can generate the event error **BadWindow**.

¦ Copyright IBM Corp. 1985, 1991

2.4.174 XGetImage

XImage *__XGetImage__(*display, drawable, x, y, width, height, plane_mask, format*)*
Display **display*;*
Drawable *drawable*;*
int *x, y*;*
unsigned int *width, height*;*
long *plane_mask*;*
int *format*;*

display	Specifies the connection to the X Server.
drawable	Specifies the drawable.
x, y	Specifies the *x* and *y* coordinates. These coordinates, which are relative to the origin of the drawable, define the upper-left corner of the rectangle.
width, height	Specifies the *width* and *height* which define the dimensions of the rectangle of the subimage.
plane_mask	Specifies the plane mask.
format	Specifies the format for the image.

XGetImage returns the contents of a rectangle in the specified drawable on the display. It returns the **XImage** structure. (See "Transferring Images Between Client and Server" in topic 1.9.3 for information on this structure.)

If *format* is **XYPixmap**, the **XGetImage** transmits the bit planes passed to the *plane_mask* argument only. If the *plane_mask* argument requests only a subset of the planes of the display, the depth of the image returned will be the number of planes requested.

If *format* is **ZPixmap**, **XGetImage** transmits as zero the bits in all planes not specified in the *plane_mask* argument.

XGetImage performs no range checking on the values in *plane_mask* and ignores extraneous bits. It returns the depth of the image to the depth member of the **XImage** structure which is the depth of the image specified when the drawable was created.

If the drawable is a pixmap, the specified rectangle must be contained entirely within the pixmap.

If the drawable is a window, it must be a mapped window. If it does not have inferiors or overlapping windows, the specified rectangle of the window will be fully visible on the screen and contained entirely within the window. The borders of the window can be included with this request.

If the window has a *backing-store*, the *backing-store* contents are returned for obscured regions of the window. Otherwise, the return contents of such obscured regions are undefined. The returned contents of visible regions of inferiors of different depth than the specified window are also undefined.

XGetImage can generate the event errors **BadDrawable**, **BadValue**, and **BadMatch**.

¦ Copyright IBM Corp. 1985, 1991

2.4.175 XGetInputFocus

XGetInputFocus(*display, focus_return, revert_to_return*)
Display **display*;
Window **focus_return*;
int **revert_to_return*;

display Specifies the connection to the X Server.

focus_return Returns the focus window ID, **PointerRoot**, or **None**. See **XSetInputFocus** in this chapter.

revert_to_return Returns the current focus state which can be **RevertToParent**, **RevertToPointerRoot**, or **RevertToNone**. See **XSetInputFocus** in this chapter.

XGetInputFocus returns the focus window ID and the current focus state.

¦ Copyright IBM Corp. 1985, 1991

2.4.176 XGetKeyboardControl

XGetKeyboardControl(*display, values_return*)
Display **display*;
XKeyboardState **values_return*;

display Specifies the connection to the X Server.

values_return Returns the current keyboard parameter in the specified **XKeyboardState** structure.

XGetKeyboardControl returns the current control values for the keyboard to the **XKeyboardState** structure. See "Manipulating Keyboard Settings" in topic 1.10.4 for more information on the **XKeyboardState** data structure.

¦ Copyright IBM Corp. 1985, 1991

2.4.177 XGetKeyboardMapping

KeySym *XGetKeyboardMapping(*display, first_keycode_wanted, keycode_count, keysyms_per_keycode_return*)
Display **display*;
KeyCode *first_keycode_wanted*;
int *keycode_count*;
int **keysyms_per_keycode_return*;

display	Specifies the connection to the X Server.
first_keycode_wanted	Specifies the first keycode to be returned.
keycode_count	Specifies the number of keycodes to be returned.
keysyms_per_keycode_return	Returns the number of keysyms per keycode.

XGetKeyboardMapping returns the symbols for the specified number of keycodes starting with the value in *first_keycode_wanted* which must be greater than or equal to *min_keycode* as returned in the **Display** structure at connection setup.

In addition, the following expression must be less than or equal to *max_keycode* as returned in the **Display** structure at connection setup.

```
first_keycode_wanted + keycode_count - 1
```

The number of elements in the keysyms list is:

```
keycode_count * keysyms_per_keycode_return
```

Then, **KeySym** number **N**, counting from zero, for keycode **K** has an index, counting from zero, of the following in **KeySym**:

```
(K - first_keycode_wanted) *  keysyms_per_keycode_return + N
```

The *keysyms_per_keycode_return* value is chosen arbitrarily by the X Server as large enough to report all requested symbols. A special **KeySym** value of **NoSymbol** is used to fill in unused elements for individual keycodes.

Use **XFree** to free the storage returned by **XGetKeyboardMapping**.

XGetKeyboardMapping can generate the event error **BadValue**.

¦ Copyright IBM Corp. 1985, 1991

2.4.178 XGetModifierMapping

XModifierKeymap *__XGetModifierMapping__(*display*)
Display **display*;

display Specifies the connection to the X Server.

XGetModifierMapping returns a newly created **XModifierKeymap** structure that contains the keycodes being used as modifiers. If only zero values appear in the set for any modifier, that modifier is disabled.

The **XModifierKeymap** structure should be freed with **XFreeModifierMapping** after this function is completed.

See "Manipulating Keyboard Encoding" in topic 1.10.5 for information on the **XModifierKeymap** data structure.

¦ Copyright IBM Corp. 1985, 1991

2.4.179 XGetMotionEvents

XTimeCoord *__XGetMotionEvents__(*display, window, start, stop, nevents_return*)
Display **display*;
Window *window*;
Time *start, stop*;
int **nevents_return*;

display	Specifies the connection to the X Server.
window	Specifies the window ID of the window whose associated pointer motion events are to be retrieved.
start, stop	Specifies the time interval in which the events are returned from the motion history buffer in a timestamp, which is expressed in milliseconds, or **CurrentTime**.
nevents_return	Returns the number of events from the motion history buffer.

XGetMotionEvents returns all events in the motion history buffer that fall between the specified start and stop times, inclusive, and that have coordinates within the specified window (including the borders) at the window's present placement.

If the start time is later than the stop time or if the start time is in the future, no events are returned. If the stop time is in the future, it is equivalent to specifying **CurrentTime**.

The return type for this function is a structure defined as:

```
typedef struct {
     Time time;
     unsigned short x, y;
} XTimeCoord;
```

The members in this structure are:

The *time* member which is set to the time in milliseconds.

The *x* and *y* members which are set to the coordinates of the pointer. These coordinates are reported relative to the origin of the specified window.

Use **XFree** to free the data returned.

XGetMotionEvents can generate the event error **BadWindow**.

¦ Copyright IBM Corp. 1985, 1991

2.4.180 XGetNormalHints

Status XGetNormalHints(*display, window, hints_return*)
Display **display*;
Window *window*;
XSizeHints **hints_return*;

display Specifies the connection to the X Server.

window Specifies the window ID.

hints_return Returns the sizing hints for the window in its normal state.

XGetNormalHints returns the size hints for a window in its normal state in its last argument. If it succeeds, it returns a non-zero. If it fails, it returns a zero. This function can fail if the application specified no normal size hints for this window. (See "Setting and Getting Window Manager Sizing Hints" in topic 1.12.5.)

This routine can generate the event error **BadWindow**.

¦ Copyright IBM Corp. 1985, 1991

2.4.181 XGetPixel

```
unsigned long XGetPixel(ximage, x, y)
      XImage *ximage;
      int x;
      int y;
```

ximage Specifies a pointer to the image.

x, y Specifies the *x* and *y* coordinates, which are relative to the origin of the image. These coordinates define the upper-left corner of the image.

XGetPixel gets a pixel value in an image. It returns the pixel from the specified image in normal format. (The least-significant byte of the long is the least-significant byte of the pixel.) See "Transferring Images Between Client and Server" in topic 1.9.3 for information about the **XImage** structure.

¦ Copyright IBM Corp. 1985, 1991

2.4.182 XGetPointerControl

XGetPointerControl(*display, accel_numerator_return, accel_denominator_return, threshold_return*)
Display **display*;
int **accel_numerator_return, *accel_denominator_return*;
int **threshold_return*;

display	Specifies the connection to the X Server.
accel_numerator_return	Returns the numerator for the acceleration multiplier.
accel_denominator_return	Returns the denominator for the acceleration multiplier.
threshold_return	Returns the acceleration threshold.

XGetPointerControl returns the current acceleration multiplier and acceleration threshold of the pointer.

¦ Copyright IBM Corp. 1985, 1991

2.4.183 XGetPointerMapping

int XGetPointerMapping(*display, map, nmap*)
Display **display*;
unsigned char *map[];*
int *nmap*;

display Specifies the connection to the X Server.

map Specifies the mapping list.

nmap Specifies the number of items in the mapping list.

XGetPointerMapping returns the current mapping of the pointer. Elements in the list are indexed starting from one. The number of items in the list is the number of physical buttons. The nominal mapping for a pointer is the identity mapping:

```
map[i]=i
```

¦ Copyright IBM Corp. 1985, 1991

2.4.184 XGetScreenSaver

XGetScreenSaver(*display, timeout_return, interval_return, prefer_blanking_retu*
allow_exposures_return)
Display **display*;
int **timeout_return, *interval_return*;
int **prefer_blanking_return*;
int **allow_exposures_return*;

display	Specifies the connection to the X Server.
timeout_return	Returns the timeout, in minutes, until the screen saver turns on.
interval_return	Returns the interval between screen saver invocations.
prefer_blanking_return	Returns the current screen blanking preference.
allow_exposures_return	Returns the current screen save control value.

XGetScreenSaver turns the screen saver on or off.

If exposure events are allowed or the screen can be regenerated without sending exposure events to clients, the screen is tiled with the root window background tile at randomly re-origined intervals. Otherwise, the state of the screen does not change and the screen saver is not activated. The screen saver is deactivated and all screen states are restored at the next keyboard or pointer input or the next call to **XForceScreenSaver** with mode **ScreenSaverReset**.

Options for *prefer_blanking_return* are **DontPreferBlanking**, **PreferBlanking**, and **DefaultBlanking**.

Options for *allow_exposures_return* are **DontAllowExposures**, **AllowExposures**, and **DefaultExposures**.

See **XSetScreenSaver** in this chapter.

¦ Copyright IBM Corp. 1985, 1991

2.4.185 XGetSelectionOwner

Window XGetSelectionOwner(*display, selection*)
Display **display*;
Atom *selection*;

display Specifies the connection to the X Server.

selection Specifies the selection atom to be returned.

XGetSelectionOwner returns the selection owner. It returns the window ID associated with the window that currently owns the specified selection. If no selection was specified or no owner exists, this function returns **None**.

XGetSelectionOwner can generate the event error **BadAtom**.

¦ Copyright IBM Corp. 1985, 1991

2.4.186 XGetSizeHints

Status XGetSizeHints(*display, window, hints_return, property*)
Display **display*;
Window *window*;
XSizeHints **hints_return*;
Atom *property*;

display Specifies the connection to the X Server.

window Specifies the window ID.

hints_return Returns the size hints.

property Specifies the property atom.

XGetSizeHints returns the **XSizeHints** structure for the named property and the specified window. This function is used by **XGetNormalHints** and **XGetZoomHints**. It also can be used to retrieve the value of any property of WM_SIZE_HINTS. Thus, it may be useful if other properties of that type are defined.

XGetSizeHints returns a non-zero status if a size hint was defined. It returns a zero if a size hint was undefined. See "Setting and Getting Window Manager Sizing Hints" in topic 1.12.5.

XGetSizeHints can generate event errors **BadAtom** and **BadWindow**.

¦ Copyright IBM Corp. 1985, 1991

2.4.187 XGetStandardColormap

Status XGetStandardColormap(*display, window, cmap_return, property*)
Display **display*;
Window *window*;
XStandardColormap **cmap_return*;
Atom *property*;

display Specifies the connection to the X Server.

window Specifies the window ID.

cmap_return Returns the colormap associated with the specified atom.

property Specifies the property atom.

XGetStandardColormap returns the colormap definition associated with the atom supplied as the *property* argument.

For example, to get the standard gray-scale colormap for a display, use **XGetStandardColormap** with the following syntax:

```
XGetStandardColormap(display, DefaultRootWindow(display),
                     cmap, RGB_GRAY_MAP);
```

This colormap can be used to convert RGB values into pixel values. For example, given an **XStandardColormap** structure and floating point RGB coefficients in the range 0.0 to 1.0, compose pixel values with the following **C** expression:

```
pixel = base_pixel + ((unsigned long)(0.5 + r * red_max)) * red_mult
                   + ((unsigned long)(0.5 + g * green_max)) * green_mult
                   + ((unsigned long)(0.5 + b * blue_max)) * blue_mult;
```

Using addition rather than logical OR for composing pixel values permits allocations where the RGB value is not aligned to bit boundaries. See "Using Standard Colormaps" in topic 1.7.4 for information on colormaps.

XGetStandardColormap can generate event errors **BadAtom** and **BadWindow**.

¦ Copyright IBM Corp. 1985, 1991

2.4.188 XGetSubImage

XImage *XGetSubImage(*display, drawable, x, y, width, height, plane_mask, forma*
dest_image, dest_x, dest_y)
Display **display*;
Drawable *drawable*;
int *x, y*;
unsigned int *width, height*;
unsigned long *plane_mask*;
int *format*;
XImage **dest_image*;
int *dest_x, dest_y*;

display Specifies the connection to the X Server.

drawable Specifies the drawable.

x, y Specifies the *x* and *y* coordinates, which are relative to the origin of the specified drawable. These coordinates define the upper-left corner of the rectangle.

width, height Specifies the *width* and *height* which define the dimensions of rectangle of the subimage.

plane_mask Specifies the plane mask.

format Specifies the format (**XYPixmap** or **ZPixmap**) for the image.

dest_image Specifies the destination image.

dest_x, dest_y Specifies the *x* and *y* coordinates of the destination rectangle. These coordinates, which are relative to the origin of the rectangle, specify the upper-left corner of the destination rectangle. These coordinates determine where the subimage is within the destination image.

XGetSubImage updates the destination image (*dest_image*) with the specified subimage. The depth of the destination image must be the same as that of the specified drawable. If the specified subimage does not fit at the specified location on the destination image, the right and bottom edges are clipped. See "Transferring Images Between Client and Server" in topic 1.9.3 for information on images.

If *format* is **XYPixmap**, it transmits only the bit planes passed to the *plane_mask* argument.

If *format* is **ZPixmap**, it transmits the bits (as zero) in all planes not specified in the *plane_mask* argument.

XGetSubImage performs no range checking on the values in *plane_mask* and ignores extraneous bits. It updates only the data fields in the destination data **XImage** structure.

If the drawable is a window, it must be a mapped window. If there are no inferiors or overlapping windows, the specified rectangle of the window is fully visible on the screen.

If the window has a *backing-store*, then the *backing-store* contents are returned for obscured regions of the window. Otherwise, the contents returned for obscured regions of the window are undefined.

¦ Copyright IBM Corp. 1985, 1991

The contents returned of visible regions of inferiors with a depth different than the depth of the specified window are also undefined.

XGetSubImage can generate the event errors **BadDrawable**, **BadGC**, **BadMatch**, and **BadValue**.

¦ Copyright IBM Corp. 1985, 1991

2.4.189 XGetTransientForHint

Status XGetTransientForHint(*display, window, prop_window_return*)
Display **display*;
Window *window*;
Window **prop_window_return*;

display	Specifies the connection to the X Server.
window	Specifies the window ID.
prop_window_return	Returns the WM_TRANSIENT_FOR property of the specified window.

XGetTransientForHint obtains the WM_TRANSIENT_FOR property for the specified window.

This routine can generate the event error **BadWindow**.

¦ Copyright IBM Corp. 1985, 1991

2.4.190 XGetVisualInfo

XVisualInfo *XGetVisualInfo(*display, vinfo_mask, vinfo_template, nitems_return*)
Display **display*;
long *vinfo_mask*;
XVisualInfo **vinfo_template*;
int **nitems_return*;

display	Specifies the connection to the X Server.
vinfo_mask	Specifies the visual mask value.
vinfo_template	Specifies the visual attributes for matching the visual structures. See "Determining the Appropriate Visual" in topic 1.5.2.
nitems_return	Returns the number of matching visual structures.

XGetVisualInfo gets a list of information structures that match a template specified in the *vinfo_template* argument. If no visual structures match the template, **XGetVisualInfo** returns a **NULL**.

¦ Copyright IBM Corp. 1985, 1991

2.4.191 XGetWindowAttributes

Status XGetWindowAttributes(*display, window, window_attributes_return*)
Display **display*;
Window *window*;
XWindowAttributes **window_attributes_return*;

display	Specifies the connection to the X Server.
window	Specifies the window ID for the window whose current attributes are to be obtained.
window_attributes_return	Returns the attributes in the **XWindowAttributes** structure of the specified window.

XGetWindowAttributes obtains the current attributes for a specified window. It returns the attributes to an **XWindowAttributes** data structure. See "Obtaining Window Information" in topic 1.6.2.

XGetWindowAttributes can generate the event error **BadWindow**.

¦ Copyright IBM Corp. 1985, 1991

2.4.192 XGetWindowProperty

int XGetWindowProperty(*display, window, property, long_offset, long_length, de*
*req_type, actual_type_return, actual_format_*
nitems_return,bytes_after_return, prop_retur
Display **display*;
Window *window*;
Atom *property*;
long *long_offset*, *long_length*;
Bool *delete*;
Atom *req_type*;
Atom **actual_type_return*;
int **actual_format_return*;
unsigned long **nitems_return*;
long **bytes_after_return*;
unsigned char ***prop_return*;

display	Specifies the connection to the X Server.
window	Specifies the window ID for the window whose atom type and property format is to be obtained.
property	Specifies the property atom.
long_offset	Specifies the offset (in 32-bit quantities) in the specified property where data will be retrieved.
long_length	Specifies the length (in 32-bit multiples) of the data to be retrieved.
delete	Specifies a Boolean value that determines if the property is to be deleted from the window. It can be set to **True** or **False**.
req_type	Specifies an atom identifier or **AnyPropertyType** associated with the property type.
actual_type_return	Returns the atom identifier that defines the type of the property.
actual_format_return	Returns the actual format (in 8-bit, 16-bit, or 32-bit) of the property.
nitems_return	Returns the actual number of items transferred.
bytes_after_return	Returns the number of bytes remaining in the property if a partial read was performed.
prop_return	Returns a pointer to the data in the specified format.

XGetWindowProperty obtains the atom type and property format for a specified window. It sets the return arguments according to the following:

If the specified property does not exist for the specified window **XGetWindowProperty** returns **None** to the *actual_type_return* argument and zero to the *actual_format_return* and *bytes_after_return* arguments. The *nitems_return* argument is empty. The *delete* argument is ignored.

¦ Copyright IBM Corp. 1985, 1991

If the specified property exists, but the property type does not matc the specified type, **XGetWindowProperty** returns the actual property type to the *actual_type_return* argument. It returns the actual property format (never zero) to the *actual_format_return* argument. It also returns the property length in bytes (even if the *actual_format_return* is 16-bit or 32-bit) to the *bytes_after_return* argument. It ignores the delete argument. The *nitems_return* argument is empty.

If the specified property exists and the *req_type* is set to **AnyPropertyType** or if the specified type matches the actual property type, **XGetWindowProperty** returns the actual property type to the *actual_type_return* argument and returns the actual property format (never zero) to the *actual_format_return* argument. **XGetWindowProperty** also returns a value to the *bytes_after_return* and *nitems_return* arguments by defining the following values:

```
N = actual length of the stored property in bytes
I = 4 * long_offset
T = N - I
L = MINIMUM(T, 4 * long_length)
A = N - (I + L)
```

The value returned starts at byte index **I** in the property (indexing from zero). The length in bytes is **L**. If the value for *long_offset* makes **L** negative, an error is generated.

The *bytes_after_return* is **A**, giving the number of trailing unread bytes in the stored property.

If *delete* is **True** and *bytes_after_return* is zero, the function deletes the property from the window and generates a **PropertyNotify** event on the window. See "Processing PropertyNotify Events" in topic 1.11.18.2.

XGetWindowProperty can generate the event errors **BadAtom**, **BadValue**, and **BadWindow**.

¦ Copyright IBM Corp. 1985, 1991

2.4.193 XGetWMHints

XWMHints *__XGetWMHints__(*display, window*)
Display **display*;
Window *window*;

display Specifies the connection to the X Server.

window Specifies the window ID.

XGetWMHints reads the value of the window manager hints atom. If it succeeds, it returns a pointer to a **XWMHints** structure. See "Setting and Getting Window Manager Hints" in topic 1.12.4 for more information.

If **XGetWMHints** fails, it returns **NULL**. This function fails if a WM_HINTS property was set for the specified window.

Free the memory created by this function after it has completed.

XGetWMHints can generate the event error **BadWindow**.

¦ Copyright IBM Corp. 1985, 1991

2.4.194 XGetZoomHints

Status XGetZoomHints(*display, window, zhints_return*)
Display **display*;
Window *window*;
XSizeHints **zhints_return*;

display Specifies the connection to the X Server.

window Specifies the window ID.

zhints_return Returns the zoom hints.

XGetZoomHints returns the size hints for a window in its normal state. It returns these hints in its last argument. If it succeeds, it returns a non-zero. If it fails, it returns a zero. **XGetZoomHints** can fail if the application did not specify zoom size hints for this window. See "Setting and Getting Window Manager Sizing Hints" in topic 1.12.5.

XGetZoomHints can generate the event error **BadWindow**.

¦ Copyright IBM Corp. 1985, 1991

2.4.195 XGrabButton

XGrabButton(*display, button_grab, modifiers, grab_window, owner_events,*
event_mask, pointer_mode, keyboard_mode, confine_to, cursor
Display **display*;
unsigned int *button_grab*;
unsigned int *modifiers*;
Window *grab_window*;
Bool *owner_events*;
unsigned int *event_mask*;
int *pointer_mode, keyboard_mode*;
Window *confine_to*;
Cursor *cursor*;

display	Specifies the connection to the X Server.
button_grab	Specifies the pointer button to be grabbed when the specified modifier keys are down.
modifiers	Specifies the set of keymasks. This mask is the bitwise inclusive OR of valid keymask bits.
grab_window	Specifies the window ID of the window to be grabbed.
owner_events	Specifies if the pointer events are to be reported normally or with respect to the grab window if selected by the event mask.
event_mask	Specifies what pointer events are reported to the client. This a bitwise inclusive OR of valid pointer event mask bits.
pointer_mode	Controls further processing of pointer events.
keyboard_mode	Controls further processing of keyboard events.
confine_to	Specifies the window ID in which to confine the pointer or **None** if the pointer is not to be confined.
cursor	Specifies the cursor to be displayed during the grab.

XGrabButton establishes a passive grab. Consequently, in the future, if the pointer is not grabbed and the specified button is pressed when the specified modifier keys are down, (and no other buttons or modifier keys are down), and, the *grab_window* contains the pointer and, the *confine_to* window, if any, is viewable and, the passive grab on the same button/key combination does not exist on any ancestor of the grab window, then, the pointer is actively grabbed.

The active grab is terminated automatically when all buttons are released (independent of the state of the modifier keys). All modifiers specified do not need to have currently assigned keycodes. A button of **AnyButton** is equivalent to issuing the request for all possible buttons. Otherwise, it is not required that the specified button currently be assigned to a physical button.

The variable *modifiers* can be set to include the following valid keymask bits:

ShiftMask **Mod1Mask** **Mod4Mask**

¦ Copyright IBM Corp. 1985, 1991

LockMask	**Mod2Mask**	**Mod5Mask**
ControlMask	**Mod3Mask**	

This variable can also be set to **AnyModifier**, which is equivalent to issuing the grab request for all possible modifier combinations (including the combination of no modifiers).

If *owner_events* is:

True, a key event is reported to this client normally.
False, all key events are reported with respect to *grab_window*.

The *event_mask* can be set to one of the following valid pointer event mask bits:

ButtonPressMask	**PointerMotionHintMask**	**Button1MotionMask**
ButtonReleaseMask	**ButtonMotionMask**	**Button2MotionMask**
EnterWindowMask	**KeymapStateMask**	**Button3MotionMask**
LeaveWindowMask		**Button4MotionMask**
PointerMotionMask		**Button5MotionMask**

Both *pointer_mode* and *keyboard_mode* can be **GrabModeSync** or **GrabModeAsync**.

XGrabButton fails and generates an error if another client issues this function with the same button/key combination on the same window.

XGrabButton fails, does not establish a grab, and and generates an error if **AnyModifier** or **AnyButton** is used with a conflicting grab.

XGrabButton has no effect on an active grab.

XGrabButton can generate the event errors **BadAccess**, **BadAlloc**, **BadCursor**, **BadValue**, and **BadWindow**

¦ Copyright IBM Corp. 1985, 1991

2.4.196 XGrabKey

XGrabKey(*display, keycode, modifiers, grab_window, owner_events, pointer_mode, keyboard_mode*)
Display **display*;
int *keycode*;
unsigned int *modifiers*;
Window *grab_window*;
Bool *owner_events*;
int *pointer_mode, keyboard_mode*;

display	Specifies the connection to the X Server.
keycode	Specifies the keycode or **AnyKey** to the specific key to be grabbed.
modifiers	Specifies the set of keymasks. This is a set of bitwise inclusive OR of valid keymask bits.
grab_window	Specifies the window ID of the window associated with the keys to be grabbed.
owner_events	Specifies a Boolean value.
pointer_mode	Controls further processing of pointer events.
keyboard_mode	Controls further processing of keyboard events.

XGrabKey establishes a passive grab on the keyboard. Consequently, if the specified key, which can be a modifier key, is pressed when the specified modifier keys are down, and, no other keys are down, and, the grab window is an ancestor of or is the focus window or, the grab window is a descendent of the focus window and, it contains the pointer, these constraints are not satisfied for any ancestor of *grab_window*, then, the keyboard is actively grabbed, and the last-keyboard-grab time is set to the time at which the key was pressed (as transmitted in the **KeyPress** event), and, the **KeyPress** event is reported.

The active grab is terminated automatically when the specified key has been released (independent of the state of the modifier keys).

The variable *modifiers* can be set to include the following valid keymask bits:

ShiftMask	**Mod1Mask**	**Mod4Mask**
LockMask	**Mod2Mask**	**Mod5Mask**
ControlMask	**Mod3Mask**	

The *modifier* argument can be set to **AnyModifier**, which is equivalent to issuing the grab key request for all possible modifier combinations (including no modifiers combination). It is not required that all modifiers specified have currently assigned keycodes. Specifying **AnyKey** is equivalent to issuing the request for all possible keycodes. Otherwise, the key must be in the range specified by *min_keycode* and *max_keycode* in the connection setup.

Both *pointer_mode* and *keyboard_mode* can be set to **GrabModeSync** or **GrabModeAsync**.

XGrabKey fails and an error is generated if another client issues this

¦ Copyright IBM Corp. 1985, 1991

function with the same key combination on the same window.

When using **AnyModifier** or **AnyKey**, **XGrabKey** fails, no grabs are established and an error is generated if another grab conflicts with this grab.

XGrabKey can generate the event errors **BadAccess**, **BadValue**, and **BadWindow**.

¦ Copyright IBM Corp. 1985, 1991

2.4.197 XGrabKeyboard

int XGrabKeyboard(*display, grab_window, owner_events, pointer_mode, keyboard_mode, time*)
Display **display*;
Window *grab_window*;
Bool *owner_events*;
int *pointer_mode, keyboard_mode*;
Time *time*;

display	Specifies the connection to the X Server.
grab_window	Specifies the window ID of the window associated with the keyboard to be grabbed.
owner_events	Specifies a Boolean value.
pointer_mode	Controls further processing of pointer events.
keyboard_mode	Controls further processing of keyboard events.
time	Specifies the time in a timestamp, which is expressed in milliseconds, or **CurrentTime**.

XGrabKeyboard actively grabs control of the main keyboard and generates **FocusIn** and **FocusOut** events. Further key events are reported only to the grabbing client. This function overrides any active keyboard grab by this client.

If *owner_events* is:

False, all generated key events are reported with respect to the *grab_window*.
True, then if a generated key event would normally be reported to this client, it is reported normally. Otherwise, the event is reported with respect to the *grab_window*.

Both **KeyPress** and **KeyRelease** events are always reported, independent of any event selection made by the client.

The *pointer_mode* argument controls further processing of pointer events. If *pointer_mode* is:

GrabModeAsync, the processing of pointer events is unaffected by activation of the grab.
GrabModeSync, the pointer, as seen by client applications, appears to freeze, and no further pointer events are generated by the server until the grabbing client issues a **XAllowEvents** call or until the keyboard grab is released.

The *keyboard_mode* argument controls further processing of the keyboard events. If *keyboard_mode* is:

GrabModeAsync, the processing of keyboard events continues normally. If the keyboard is currently frozen by this client, then processing of keyboard events is resumed.

GrabModeSync, the keyboard events, as seen by client applications, appears to freeze, and no further keyboard events are generated by the server until the grabbing client issues a **XAllowEvents** call or until

¦ Copyright IBM Corp. 1985, 1991

the keyboard is released. Actual keyboard changes are not lost while the keyboard is frozen; instead, these changes are queued for later processing.

If **XGrabKeyboard** fails, it returns one of the following:

AlreadyGrabbed	If the keyboard is actively grabbed by another client.
GrabNotViewable	If *grab_window* is not viewable.
GrabInvalidTime	If the specified time is earlier than the last-keyboard-grab time or later than the current X Server time. Otherwise, the last-keyboard-grab time is set to the specified time and **CurrentTime** is replaced by the current X Server time.
GrabFrozen	If the keyboard is frozen by an active grab of another client.

XGrabKeyboard can generate the event errors **BadValue** and **BadWindow**.

¦ Copyright IBM Corp. 1985, 1991

2.4.198 XGrabPointer

int XGrabPointer(*display, grab_window, owner_events, event_mask, pointer_mode, keyboard_mode, confine_to, cursor, time*)
Display **display*;
Window *grab_window*;
Bool *owner_events*;
unsigned int *event_mask*;
int *pointer_mode, keyboard_mode*;
Window *confine_to*;
Cursor *cursor*;
Time *time*;

display	Specifies the connection to the X Server.
grab_window	Specifies the window ID of the window relative to which events are reported while it is grabbed.
owner_events	Specifies if the pointer events are to be reported normally or with respect to the grab window if selected by the event mask.
event_mask	Specifies what pointer events are reported to the client. This is a bitwise inclusive OR of valid pointer event mask bits.
pointer_mode	Controls further processing of pointer events.
keyboard_mode	Controls further processing of keyboard events.
confine_to	Specifies the window to confine the pointer in or **None** if the pointer is not to be confined.
cursor	Specifies the cursor to be displayed during the grab.
time	Specifies the time in a timestamp, which is expressed in milliseconds, or **CurrentTime**.

XGrabPointer actively grabs control of the pointer and returns **GrabSuccess** if the grab was successful. Further pointer events are reported only to the grabbing client. This function overrides any active pointer grab by this client.

XGrabPointer generates **EnterNotify** and **LeaveNotify** events. See "Processing Window Entry or Exit Events" in topic 1.11.7.

If *owner_events* is:

False, all generated pointer events are reported with respect to the *grab_window*. These events are reported only if selected by *event_mask*. Unreported events are discarded.
True, and if a generated pointer event would be reported to this client normally, it is reported normally. Otherwise, the event is reported with respect to the *grab_window* and is reported only if selected by the *event_mask*. Unreported events are discarded.

The *event_mask* can be set to one of the following:

ButtonPressMask	**PointerMotionHintMask**	**Button1MotionMask**
ButtonReleaseMask	**ButtonMotionMask**	**Button2MotionMask**

¦ Copyright IBM Corp. 1985, 1991

EnterWindowMask	**KeymapStateMask**	**Button3MotionMask**
LeaveWindowMask		**Button4MotionMask**
PointerMotionMask		**Button5MotionMask**

The *pointer_mode* controls further processing of pointer events. If *pointer_mode* is:

GrabModeAsync, processing of pointer events continues normally. If the pointer is currently frozen by this client, the processing of events for the pointer is resumed.
GrabModeSync, the pointer, as seen by client applications, appears to freeze, and no further pointer events are generated by the X Server until the grabbing client calls **XAllowEvents** or until the pointer grab is released. Actual pointer changes are not lost while the pointer is frozen; they are simply queued for later processing.

The *keyboard_mode* controls further processing of main keyboard events. If the *keyboard_mode* is:

GrabModeAsync, keyboard event processing is unaffected by activation of the grab.
GrabModeSync, the keyboard, as seen by client applications, appears to freeze and no further keyboard events are generated by the X Server until the grabbing client calls **XAllowEvents** or until the pointer grab is released. Actual keyboard changes are not lost while the pointer is frozen; they are simply queued for later processing.

If *cursor* is specified, it is displayed regardless of which window contains the pointer. If *cursor* is not specified, the normal cursor for that window is displayed when the pointer is in the *grab_window* or one of its subwindows.

If a *confine_to* window is specified, the pointer is restricted to that window. If the pointer is not in the *confine_to* window initially, it is warped automatically to the closest edge of that window before the grab activates. Enter or leave events are generated normally. If the *confine_to* window is reconfigured consequently, the pointer is warped automatically to contain it within the window.

The *time* argument helps avoid certain situations that may occur. For example, if two applications that normally grab the pointer when clicked on have a specified timestamp, then the second application may grab the pointer successfully, while the first application is notified that the pointer was grabbed before its request was processed.

If **XGrabPointer** fails, it returns one of the following:

GrabNotViewable	If *grab_window* or *confine_to* window is not viewable.
AlreadyGrabbed	If the pointer is actively grabbed by another client.
GrabFrozen	If the pointer is frozen by an active grab of another client.
GrabInvalidTime	If the specified time is earlier than the last-pointer-grab time or later than the current X Server time. Otherwise, the last-pointer-grab time is set to the specified time and **CurrentTime** is replaced by the current X Server time.

¦ Copyright IBM Corp. 1985, 1991

XGrabPointer can generate the event errors **BadCursor**, **BadValue**, and **BadWindow**.

¦ Copyright IBM Corp. 1985, 1991

2.4.199 XGrabServer

XGrabServer(*display*)
Display **display*;

display Specifies the connection to the X Server.

XGrabServer disables processing of requests on all connections except the one that it arrived on. It performs a closedown. The X Server should be grabbed only for brief periods because no processing of requests or closedowns on any other connections occur while it is grabbed.

¦ Copyright IBM Corp. 1985, 1991

2.4.200 XIfEvent

XIfEvent(*display, event_return, predicate, arg*)
Display **display*;
XEvent **event_return*;
Bool (**predicate*)();
char **arg*;

display	Specifies the connection to the X Server.
event_return	Copies the structure of the matched event into this client-supplied structure.
predicate	Specifies the procedure which is called to determine if the next event in the queue matches the one event specified in the *arg* argument.
arg	Specifies the user-supplied argument passed to the predicate procedure.

XIfEvent flushes the output buffer if it blocks waiting for an event. When the predicate procedure returns, **XIfEvent** removes the event from the queue and copies the event structure into the client-supplied **XEvent** structure.

XIfEvent completes only when the specified predicate procedure returns **True**, a non-zero value. This return value indicates that an event on the queue matched the specified event. This predicate procedure also is called when an event is added to the queue.

¦ Copyright IBM Corp. 1985, 1991

2.4.201 XInsertModifiermapEntry

XModifierKeymap *__XInsertModifiermapEntry__(*modmap, keycode, modifier*)*
XModifierKeymap **modmap*;
KeyCode *keycode*;
int *modifier*;

modmap Specifies a pointer to the **XModifierKeymap** structure.

keycode Specifies the keycode.

modifier Specifies the modifier.

XInsertModifiermapEntry adds the specified keycode to the set that controls the specified modifier. It returns the result to the **XModifierKeymap** structure. See "Manipulating Keyboard Encoding" in topic 1.10.5.

¦ Copyright IBM Corp. 1985, 1991

2.4.202 XInstallColormap

XInstallColormap(*display, cmap*)
Display **display*;
Colormap *cmap*;

display Specifies the connection to the X Server.

cmap Specifies the colormap ID.

XInstallColormap installs the specified colormap for its associated screen. All windows associated with this colormap immediately display with true colors. The windows associated with the specified colormaps were created with **XCreateWindow** or **XCreateSimpleWindow**.

The X Server obtains the colormap from a ***required list***, which is an ordered list containing a subset of the installed colormaps. If the specified colormap is not an installed colormap, the X Server generates a **ColormapNotify** event on every window that has the *cmap* as the colormap resource ID. (See "Processing Colormap State Notification Events" in topic 1.11.17.)

In addition, the X Server generates a **ColormapNotify** event on every window that has *cmap* as the colormap resource ID for every other colormap installed or uninstalled as a result of **XInstallColormap**. The colormaps that get installed or uninstalled are server-dependent, except that the required list must remain installed.

This routine can generate the event error **BadColor**.

¦ Copyright IBM Corp. 1985, 1991

2.4.203 XInternAtom

Atom XInternAtom(*display*, *atom_name*, *only_if_exists*)
Display **display*;
char **atom_name*;
Bool *only_if_exists*;

display Specifies the connection to the X Server.

atom_name Specifies the name associated with the atom to be returned. (See "Using Properties and Atoms" in topic 1.6.4.)

only_if_exists Specifies a Boolean value that indicates if **XInternAtom** creates the atom.

XInternAtom returns an atom for a specified name. It returns the atom identifier associated with the string passed to the *atom_name* argument. The atom remains defined even after the client that defined it has gone away. The atom becomes undefined after the last connection to the X Server closes.

If *only_if_exists* is:

True, **XInternAtom** returns the atom specified in *atom_name*.
False, **XInternAtom** creates the atom if it does not exist.

XInternAtom can return **None**.

XInternAtom can generate the event errors **BadAlloc** and **BadValue**.

¦ Copyright IBM Corp. 1985, 1991

2.4.204 XIntersectRegion

XIntersectRegion(*sra, srb, dr*)
Region *sra, srb, dr*;

sra, srb Specifies the two regions with which to perform the computation.

dr Stores the result of the computation.

XIntersectRegion computes the intersection of two regions.

¦ Copyright IBM Corp. 1985, 1991

2.4.205 XKeycodeToKeysym

KeySym XKeycodeToKeysym(*display, keycode, index_return*)
Display **display*;
KeyCode *keycode*;
int *index_return*;

display Specifies the connection to the X Server.

keycode Specifies the keycode.

index_return Returns the element of keycode vector.

XKeycodeToKeysym converts a keycode to a defined keysym. It uses internal **Xlib** tables. It returns the defined keysym for the specified keycode and the element of the keycode vector. See "Manipulating Keyboard Encoding" in topic 1.10.5.

¦ Copyright IBM Corp. 1985, 1991

2.4.206 XKeysymToKeycode

KeyCode XKeysymToKeycode(*display, keysym_kcode*)
Display **display*;
Keysym *keysym_kcode*;

display Specifies the connection to the X Server.

keysym_kcode Specifies the keysym that is to be searched for.

XKeysymToKeycode converts a keysym to the appropriate keycode. If the keysym specified is not defined for any keycode, **XKeysymToKeycode** returns a zero.

¦ Copyright IBM Corp. 1985, 1991

2.4.207 XKeysymToString

char *XKeysymToString(*keysym_str*)
KeySym *keysym_str;*

keysym_str Specifies the keysym to be converted.

XKeysymToString converts a keysym code to the name of the keysym. The string returned is in a static area and must not be modified. If the specified keysym is not defined, **XKeysymToString** returns a **NULL**.

¦ Copyright IBM Corp. 1985, 1991

2.4.208 XKillClient

XKillClient(*display, resource*)
Display **display*;
XID *resource*;

display Specifies the connection to the X Server.

resource Specifies any resource associated with the client to be destroyed.

XKillClient forces a closedown of the client that created the resource if a valid resource is specified.

If the client has terminated already in **RetainPermanent** or **RetainTemporary** mode, all of the client's resources are destroyed and the server is not reset.

If **AllTemporary** is specified, the resources of all clients that have terminated in **RetainTemporary** are destroyed and the server is reset.

XKillClient generates the event error **BadValue**.

¦ Copyright IBM Corp. 1985, 1991

2.4.209 XListFonts

char ****XListFonts**(*display, pattern, maxnames, actual_count_return*)
Display **display*;
char **pattern*;
int *maxnames*;
int **actual_count_return*;

display	Specifies the connection to the X Server.
pattern	Specifies the null-terminated string associated with the font names to be returned. Specify an asterisk (*), which indicates a wildcard on any number of characters, or a question mark (?), which indicates a wildcard on a single character.
maxnames	Specifies the maximum number of names to be in the returned list.
actual_count_return	Returns the actual number of font names.

XListFonts returns an array of font names that matches the string in the *pattern* argument. Each string is terminated by **NULL**. The maximum number of names returned in the list is dependent on the value in *maxnames*. **XListFonts** places the actual number of font names returned in *actual_count_return*. See "Manipulating Fonts" in topic 1.9. The client should call **XFreeFontNames** when this function is completed.

¦ Copyright IBM Corp. 1985, 1991

2.4.210 XListFontsWithInfo

char **__**XListFontsWithInfo**__(*display, pattern, maxnames, count_return, info_retur*
Display **display*;*
char **pattern*;*
int *maxnames*;*
int **count_return*;*
XFontStruct ***info_return*;*

display Specifies the connection to the X Server.

pattern Specifies the null-terminated string associated with the font names to be returned. Specify any string with an asterisk (*), which indicates a wildcard on any number of characters, or a question mark (?), which indicates a wildcard on a single character.

maxnames Specifies the maximum number of names to be in the returned list.

count_return Returns the actual number of matched font names.

info_return Returns a pointer to the font information.

XListFontsWithInfo returns a list of names of fonts that match the *pattern* specified and the associated font information. The list of names is limited to the number specified in *maxnames*.

To free the allocated name array, the client should call **XFreeFontNames**. To free the font information array, the client should call **XFreeFontInfo**. See "Manipulating Fonts" in topic 1.9.

¦ Copyright IBM Corp. 1985, 1991

2.4.211 XListHosts

XHostAddress ***XListHosts**(*display, nhosts_return, state_return*)
Display **display*;
int **nhosts_return*;
Bool **state_return*;

display Specifies the connection to the X Server.

nhosts_return Returns the number of hosts currently in the access control list.

state_return Returns the state (enabled or disabled) of the access control.

XListHosts returns the current access control list and the the state of the control which indicates if connection setup was enabled or disabled. It allows a client to determine what systems can make connections. It returns a pointer to a list of host structures that were allocated by the routine. (See "Controlling Host Access" in topic 1.10.6.)

Use **XFree** to free the allocated memory when it is no longer needed.

¦ Copyright IBM Corp. 1985, 1991

2.4.212 XListInstalledColormaps

Colormap *__XListInstalledColormaps__(*display, window, num_return*)
Display **display*;
Window *window*;
int **num_return*;

display Specifies the connection to the X Server.

window Specifies the window ID for the window whose screen list of currently installed colormaps is to be obtained.

num_return Returns the list of currently installed colormaps.

XListInstalledColormaps returns a list of the currently installed colormaps for the screen of the specified window. The order of the colormaps in the list is insignificant, and there is no explicit indication of the required list.

Use **XFree** to free the allocated list when it is no longer needed.

XListInstalledColormaps can generate the event error **BadWindow**.

¦ Copyright IBM Corp. 1985, 1991

2.4.213 XListProperties

Atom *XListProperties(*display, window, num_prop_return*)
Display **display*;
Window *window*;
int **num_prop_return*;

display Specifies the connection to the X Server.

window Specifies the window ID for the window whose property list is to be obtained.

num_prop_return Returns the length of the properties array.

XListProperties obtains a property list for a specified window. It returns a pointer to an array of atom properties that are defined for the specified window. (See "Using Properties and Atoms" in topic 1.6.4.)

Use **XFree** to free the allocated memory when it is no longer needed.

XListProperties can generate the event error **BadWindow**.

¦ Copyright IBM Corp. 1985, 1991

2.4.214 XLoadFont

Font XLoadFont(*display, name*)
Display **display*;
char **name*;

display Specifies the connection to the X Server.

name Specifies the font name (a null-terminated string).

XLoadFont loads the specified font and returns the font ID. If this function is unsuccessful, it generates an error. (See "Manipulating Fonts" in topic 1.9.) Use **XUnloadFont** when the font is no longer needed.

Fonts are not associated with a particular screen and can be stored as a component of any graphics context.

XLoadFont can generate the event errors **BadAlloc** and **BadName**.

¦ Copyright IBM Corp. 1985, 1991

2.4.215 XLoadQueryFont

XFontStruct *XLoadQueryFont(*display, name*)
Display **display*;
char **name*;

display Specifies the connection to the X Server.

name Specifies the font name (a null-terminated string).

XLoadQueryFont provides the most common way for accessing a font. It opens or loads the specified font and returns a pointer to the appropriate **XFontStruct** structure. **XLoadQueryFont** returns **NULL** if the specified font does not exist.

XFontStruct contains all the font information and a pointer to an array of **XCharStruct** structures for the characters contained in the font. See "Manipulating Fonts" in topic 1.9.

XLoadQueryFont can generate the event error **BadAlloc**.

¦ Copyright IBM Corp. 1985, 1991

2.4.216 XLookupColor

Status XLookupColor(*display, cmap, colorname, exact_def_return, screen_def_ret*
Display **display*;
Colormap *cmap*;
char **colorname*;
XColor *exact_def_return, **screen_def_return*;

display	Specifies the connection to the X Server.
cmap	Specifies the colormap ID.
colorname	Specifies the string of the color name for the color definition structure to be returned.
exact_def_return	Returns the exact RGB values for the color specified in *colorname*.
screen_def_return	Returns the closest RGB values provided by the hardware.

XLookupColor looks up the string of a color name for the screen associated with the *cmap* specified. It returns the color values and the closest colors available for the hardware. (See "Creating Colormaps" in topic 1.7.1.) If **XLookupColor** is:

successful, it returns a a non-zero because the color is in the RG database.
not successful, it returns a zero because the color is not in the RG database.

XLookupColor can generate the event errors **BadColor** and **BadName**.

¦ Copyright IBM Corp. 1985, 1991

2.4.217 XLookupKeysym

KeySym XLookupKeysym(*event_key, index*)
XKeyEvent **event_key*;
int *index;*

event_key Specifies the key event (**KeyPress** or **KeyRelease**) to be used.

index Specifies the index into the **KeySyms** table.

XLookupKeysym looks up the **KeySyms**. It uses a given keyboard event and the index specified to return the **KeySym** from the list that corresponds to the keycode member in the **XKeyPressedEvent** or **XKeyReleasedEvent** structures. (See "Processing Common Keyboard and Pointer Events" in topic 1.11.5.2.)

¦ Copyright IBM Corp. 1985, 1991

2.4.218 XLookupMapping

char* XLookupMapping (*event, nbytes*)
XKeyPressedEvent **event*;
int **nbytes*;

event Specifies the **KeyPress** event to be used.

nbytes Returns a pointer to the number of bytes returned in the character string or zero if no text is mapped to the event.

XLookupMapping maps events to counted character strings (an array of characters and the length; the null character is legitimate in this use). It returns a pointer to a static counted character string, which must not be modified by a client, and the number of bytes in the string.

XLookupMapping searches for the current keyboard mapping in the following order:

$XDIR/.Xkeymap
$HOME/.Xkeymap
/usr/lpp/X11/defaults/.Xkeymap

If these files are not present, **XLookupMapping** defaults to **XLookupString**.

The **.Xkeymap** file is produced by the **keycomp** command, which reads a text file of keyboard mappings. (For information about the **keycomp** command see *AIX X-Windows User's Guide*.) The keyboard mappings in this file are based on keysym values, not keycode values. Therefore, the first keysym in the list of keysyms associated with the keycode in the **XKeyPressedEvent** is used to access the **.Xkeymap** file. The directory **/usr/lpp/X11/defaults** contains the keyboard mappings for languages selected during installation of X-Windows.

XLookupMapping performs normal interpretation of shift bits (meta, shift, shift lock, and control). It supports **Alt-NumPad** and **NumLock** key processing as well as the dead key processing defined in **keycomp**.

Alt-NumPad processing begins when the first **Alt-NumPad** key is pressed and ends when either the third **Alt-NumPad** key is pressed or a non-**Alt-NumPad** key is pressed.

The final keymapping is not returned to the user until a terminating event occurs. If the terminating event is a non-**Alt-NumPad** key, then both the generated **Alt-NumPad** keycode and the string of the non-**Alt-NumPad** key is returned in a single buffer.

For this to process correctly, both the **Alt** key and the **NumPad** key (in **Alt** state) must be defined as UNBOUND in the source keymap. In addition, **XLookupMapping** tracks the **NumLock** state only if the **NumLock** key is defined as UNBOUND.

Use **strncpy** to copy the result for storage if the data must be modified. If a different keymap file is desired, use **XUseKeymap**.

Note: On the AIX PS/2 X-Windows, the **XLookupMapping** is in **/usr/lib/liboldx.a**. Use **-loldx** on the command line to link to this library.

¦ Copyright IBM Corp. 1985, 1991

2.4.219 XLookupString

int XLookupString(*event_struct, buffer_return, bytes_buffer, keysym_return, status_return*)
XKeyEvent **event_struct*;
char **buffer_return*;
int *bytes_buffer*;
KeySym **keysym_return*;
XComposeStatus **status_return*;

event_struct Specifies the key event structure (**XKeyPressedEvent** or **XKeyReleasedEvent**) to be used.

buffer_return Returns the translated characters.

bytes_buffer Specifies the length of the buffer to be returned.

keysym_return Returns the **keysym** computed from the event.

status_return Specifies the status of the processing.

XLookupString maps a key event to an ASCII string, using the modifier bits in the key event to deal with shift, lock, and control. It returns the translated string into the user's buffer. **XLookupString** also detects any rebound keysyms (see **XRebindKeysym**) and returns the specified bytes. It returns the length of the string stored in the tag buffer as its value. If the lock modifier has a **Caps_Lock** key associated with it, **XLookupString** interprets the lock modifier to perform caps lock processing.

If *status_return* returns a pointer to the **XCompose** structure, which contains compose key state information, key processing will take place. Otherwise, *status_return* is **NULL**.

¦ Copyright IBM Corp. 1985, 1991

2.4.220 XLowerWindow

XLowerWindow(*display, window*)
Display **display*;
Window *window*;

display Specifies the connection to the X Server.

window Specifies the window ID of the window to be lowered.

XLowerWindow lowers the specified window to the bottom of the stack so that it does not obscure any sibling windows. If the windows are regarded as overlapping sheets of paper stacked on a desk, then lowering a window is analogous to moving the sheet to the bottom of the stack but leaving its *x* and *y* location on the desk constant. Lowering a mapped window will generate exposure events on any formerly obscured windows.

The X Server generates a **ConfigureRequest** event and no processing is performed if the *override_redirect* attribute of the window is **False** and another client selected **SubstructureRedirectMask** on the parent window. Otherwise, the window is lowered to the bottom of the stack. See "Configuring Windows" in topic 1.5.7 for information on window attributes that can affect this routine.

XLowerWindow can generate the event error **BadWindow**.

¦ Copyright IBM Corp. 1985, 1991

2.4.221 XMapRaised

XMapRaised(*display, window*)
Display **display*;
Window *window*;

display Specifies the connection to the X Server.

window Specifies the window.

XMapRaised maps the window and all of its subwindows that have map requests. It also raises the specified window to the top of the stack. See **XMapWindow** in this chapter.

XMapRaised can generate the event error **BadWindow**.

¦ Copyright IBM Corp. 1985, 1991

2.4.222 XMapSubwindows

XMapSubwindows(*display, window*)
Display **display*;
Window *window*;

display Specifies the connection to the X Server.

window Specifies the window ID.

XMapSubwindows maps all subwindows for a specified window in top-to-bottom stacking order. The X Server generates an **Expose** event on each displayed window that has just been created.

XMapSubwindows can generate the event error **BadWindow**.

¦ Copyright IBM Corp. 1985, 1991

2.4.223 XMapWindow

XMapWindow(*display, window*)
Display **display*;
Window *window*;

display Specifies the connection to the X Server.

window Specifies the window ID.

XMapWindow maps the specified window and it subwindows with map requests. A subwindow is displayed on the screen as long as all of its ancestors are mapped and not obscured by a sibling or not clipped by an ancestor. Mapping a window that has an unmapped ancestor does not display the window, but marks it as eligible for display when the ancestor becomes mapped. Such a window is ***unviewable***. When all the ancestors of the window are mapped, the window becomes viewable and is visible on the screen if it is not obscured by a sibling or an ancestor. Mapping ancestors has no effect on the window if the window is already mapped.

The X Server generates a **MapRequest** event, and the **XMapWindow** does not map the window, if the *override_redirect* value of the **XSetWindowAttributes** structure is **False** and if another client selected **SubstructureRedirectMask** on the parent window. Otherwise, the X Server generates a **MapNotify** event and the window is mapped.

If the window becomes viewable and no earlier contents are stored, **XMapWindow** tiles the window with its background. If no background is defined for the window, the existing screen contents are not altered, and the X Server generates one or more **Expose** events.

If a *backing_store* was maintained while the window is unmapped, no **Expose** events are generated. If a *backing_store* will now be maintained, a full-window exposure is generated. Otherwise, only visible regions can be reported. Similar tiling and exposure take place for any newly viewable inferiors.

If the window is an **InputOutput** window, **XMapWindow** generates **Expose** events on each **InputOutput** window that it causes to become displayed.

If the client maps and paints the window and if the client begins processing events, the window is painted twice. To avoid this, the client should call **XSelectInput** for exposure events and map the window. Then, the client processes input events normally. The event list will include **Expose** for each window that is displayed on the screen.

The normal response of the client to an **Expose** event should be to repaint the window.

XMapWindow can generate the event error **BadWindow**.

¦ Copyright IBM Corp. 1985, 1991

2.4.224 XMaskEvent

XMaskEvent(*display, event_mask, event_return*)
Display **display*;
unsigned long *event_mask*;
XEvent **event_return*;

display Specifies the connection to the X Server.

event_mask Specifies the event mask. This mask is the bitwise inclusive OR of one or more of the valid event mask bits. See "Defining Event Masks" in topic 1.11.3.

event_return Copies the matched event's associated structure into this client-supplied structure.

XMaskEvent searches the event queue for the events associated with the specified mask. When it finds a match, it removes and copies the matched event into the specified **XEvent** structure. The other events stored in the queue are not discarded. See "Defining Event Structures" in topic 1.11.2.

If the event requested is not in the queue, **XMaskEvent** flushes the output buffer and blocks until a matched event is received.

¦ Copyright IBM Corp. 1985, 1991

2.4.225 XMatchVisualInfo

Status XMatchVisualInfo(*display, screen, depth, class, vinfo_return*)
Display **display*;
int *screen*;
int *depth*;
int *class*;
XVisualInfo **vinfo_return*;

display Specifies the connection to the X Server.

screen Specifies the screen.

depth Specifies the depth of the screen.

class Specifies the class of the screen.

vinfo_return Returns the match visual information.

XMatchVisualInfo obtains the visual information for a visual that matches the specified *depth* and *class* of the screen. The exact visual chosen is undefined because there can be more than one visual that matches the specified depth and class.

If a match is found, **XMatchVisualInfo** returns **True** and the information on the visual to the *vinfo_return* argument. If a match is not found, **XMatchVisualInfo** returns **False**. For information about the **XVisualInfo** structure, see "Determining the Appropriate Visual" in topic 1.5.2.

¦ Copyright IBM Corp. 1985, 1991

2.4.226 XMoveResizeWindow

XMoveResizeWindow(*display, window, x, y, width, height*)
Display **display*;
Window *window*;
int *x, y*;
unsigned int *width, height*;

display Specifies the connection to the X Server.

window Specifies the window ID to be reconfigured.

x, y Specifies the *x* and *y* coordinates, which are relative to the parent window and which define the new position of the window.

width, height Specifies the *width* and *height* which define the interior size of the window.

XMoveResizeWindow changes the size and location of the specified window without raising it. Moving and resizing a mapped window may generate an **Expose** event on the window. Depending on the new size and location parameters, moving and resizing a window may generate exposure events on windows that the window formerly obscured.

The X Server generates a **ConfigureRequest** event and no processing is performed if the *override_redirect* attribute of the window is **False** and another client selected **SubstructureRedirectMask** on the parent window. Otherwise, the size and location of the window is changed. See "Configuring Windows" in topic 1.5.7 for information on window attributes that can affect this routine.

XMoveResizeWindow can generate the errors **BadMatch**, **BadValue**, and **BadWindow**.

XMoveWindow(*display, window, x, y*)
Display **display*;
Window *window*;
int *x, y*;

display Specifies the connection to the X Server.

window Specifies the window ID of the window to be moved.

x, y Specifies the *x* and *y* coordinates which define the new location of the top-left pixel of the border of the window or the window itself, if it has no border.

XMoveWindow moves the specified window to the location specified by the *x* and *y* coordinates. **XMoveWindow** does not change the size or the mapping state of the window and does not raise the window. A mapped window may or may not lose its contents when it is moved depending on:

If its *background_pixmap* attribute is **ParentRelative**.
If it is obscured by non-children and has no backing store

If the contents of the window are lost, exposure events will be generated for the window and for any mapped subwindows. Moving a mapped window generates exposure events on any formerly obscured windows.

¦ Copyright IBM Corp. 1985, 1991

The X Server generates a **ConfigureRequest** event and no processing is performed if the *override_redirect* attribute of the window is **False** and another client selected **SubstructureRedirectMask** on the parent window. Otherwise, the window is moved. See "Configuring Windows" in topic 1.5.7 for information on window attributes that can affect this routine.

XMoveWindow can generate the event error **BadWindow**.

¦ Copyright IBM Corp. 1985, 1991

2.4.227 XNewModifiermap

XModifierKeymap *__XNewModifiermap**(*max_keys_per_mod*)
int *max_keys_per_mod*;

max_keys_per_mod Specifies the maximum number of keycodes assigned to any of the modifiers in the map.

XNewModifiermap returns a pointer to **XModifierKeymap** structure. Use **XFreeModifermap** to free the storage after this function has completed. See "Manipulating Keyboard Encoding" in topic 1.10.5 for more information on keyboard data structures.

¦ Copyright IBM Corp. 1985, 1991

2.4.228 XNextEvent

XNextEvent(*display, event_return*)
Display **display*;
XEvent **event_return*;

display Specifies the connection to the X Server.

event_return Copies the event structure into the client-supplied structure.

XNextEvent flushes or copies the first event from the event queue into the specified **XEvent** structure and then removes it from the queue. If the event queue is empty, **XNextEvent** flushes the output buffer and blocks until an event is received. For example, if a **CreateNotify** event is the first event in the queue, **XNextEvent** removes it and then copies the **XCreateWindowEvent** structure into the **XEvent** structure.

¦ Copyright IBM Corp. 1985, 1991

2.4.229 XNoOp

XNoOp(*display*)
Display **display*;

display Specifies the connection to the X Server.

XNoOp executes a **NoOperation** protocol request. It does not flush the output buffer.

¦ Copyright IBM Corp. 1985, 1991

2.4.230 XOffsetRegion

XOffsetRegion(*region, dx, dy*)
Region *region*;
int *dx, dy*;

region Specifies the region.

dx, dy Specifies the *x* and *y* coordinates which define the amount by which to move the specified region.

XOffsetRegion moves the specified region by a specified amount.

¦ Copyright IBM Corp. 1985, 1991

2.4.231 XOpenDisplay

Display *XOpenDisplay(*display_name*)

char **display_name;*

display_name Specifies the hardware display name, which determines the display and communications domain to be used.

XOpenDisplay opens a connection to the X Server controlling the specified display. The X Server may implement various types of access control mechanisms that allow clients to use the screens in the display.

The screen number specified in *display_name* specifies the value returned by the **DefaultScreen** macro or the **XDefaultScreen** function.

If **XOpenDisplay** is successful, it returns a pointer to a **Display** structure defined in **< X11/Xlib.h >**. After a successful call, client applications may use all the screens in the display.

If **XOpenDisplay** is not successful, it returns **NULL** and *display_name* defaults to the **DISPLAY** environment variable.

Access elements of the **Display** and **Screen** structures by using the information macros or functions. See "Using Display Functions" in topic 1.4.

¦ Copyright IBM Corp. 1985, 1991

2.4.232 XParseColor

Status XParseColor(*display, cmap, spec, screen_def_return*)
Display **display*;
Colormap *cmap*;
char **spec*;
XColor **screen_def_return*;

display	Specifies the connection to the X Server.
cmap	Specifies the colormap ID.
spec	Specifies the color name string.
screen_def_return	Returns the values used in the colormaps.

XParseColor parses color values and provides a way to create a standard user interface to color. It takes a string specification of a color, typically from a command line or **XGetDefault** option, and returns the corresponding red, green, and blue values that are suitable for a subsequent call to **XAllocColor** or **XStoreColor**.

The color can be specified as a color name (as in **XAllocNamedColor**) or as an initial sharp sign character followed by a numeric specification, in one of the following formats:

#RGB	(4 bits each)
#RRGGBB	(8 bits each)
#RRRGGGBBB	(12 bits each)
#RRRRGGGGBBBB	(16 bits each)

The R, G, and B represent single hexadecimal digits (uppercase or lowercase). When fewer than 16 bits each are specified, these bits represent the most-significant bits of the value. For example, **#3a7** is the same as **#3000a0007000**.

The colormap determines the screen to use to look up the color. For example, you can use the default colormap of the screen. See "Creating Colormaps" in topic 1.7.1.

If **XParseColor** fails, it returns a zero. It can fail for one of the following reasons:

The initial character is a sharp sign, but the string is not in th proper format.
The initial character is not a sharp sign and the color does not exis in the database of the server.

XParseColor can generate the event error **BadColor**.

¦ Copyright IBM Corp. 1985, 1991

2.4.233 XParseGeometry

int XParseGeometry(*parsestring, x_return, y_return, width_return, height_retur*
char **parsestring*;
int **x_return*, **y_return*;
int **width_return*, **height_return*;

parsestring	Specifies the string to be parsed.
x_return, y_return	Returns the xoffset and yoffset determined.
width_return, height_return	Returns the width and height determined.

XParseGeometry parses standard window geometry strings which indicate size and placement of the window. It allows you to parse strings in the following format:

```
{=}<width>x<height>{+-}<xoffset>{+-}<yoffset>
```

These items map into **XParseGeometry**. It returns a bitmask that indicates which values (width, height, xoffset, and yoffset) were found in the string. It also indicates if *x* and *y* are negative. By convention, -0 is not equal to +0, because you need to be able to say "position the window relative to the right or bottom edge."

For each value found, the corresponding argument is updated. For each value not found, the argument is left unchanged.

The bits are represented by **XValue**, **YValue**, **WidthValue**, **HeightValue**, **XNegative**, and **YNegative** which are defined in **< X11/Xutil.h >**. These values are set when they are defined or when one of the signs is set.

If the function returns **XValue** or **YValue**, place the window at the requested position. The *bwidth*, *width* and *height* sizes (typically font width and height), and any additional interior space (*xadd* and *yadd*) are used to compute the resulting size.

¦ Copyright IBM Corp. 1985, 1991

2.4.234 XPeekEvent

XPeekEvent(*display, event_return*)
Display **display*;
XEvent **event_return*;

display Specifies the connection to the X Server.

event_return Copies the event structure into this client-supplied structure.

XPeekEvent returns the first event from the event queue, but it does not remove the event from the queue. If the queue is empty, **XPeekEvent** flushes the output buffer and blocks until an event is received. Then, it copies the event into the client-supplied **XEvent** structure without removing it from the event queue. For example, if a **CreateNotify** event is the first event in the queue, **XPeekEvent** peeks at the event, but does not remove it. Then, it copies the **XCreateWindowEvent** structure into the **XEvent** structure.

Use the **QLength** macro to determine if there are events to peek at.

¦ Copyright IBM Corp. 1985, 1991

2.4.235 XPeekIfEvent

XPeekIfEvent(*display, event_return, predicate, arg*)
Display **display*;
XEvent **event_return*;
Bool (**predicate*)();
char **arg*;

display Specifies the connection to the X Server.

event_return Copies the matched event structure into this client-supplied structure.

predicate Specifies the procedure called to determine if the next event in the queue matches the one specified by the event argument.

arg Specifies the user-supplied argument passed to the predicate procedure.

XPeekIfEvent returns only when the specified predicate procedure returns a non-zero or **True** for the next event in the queue that matches the specified event.

The following predicate procedure is used:

Bool predicate(*display, event, arg*)
Display **display*;
XEvent **event*;
char **arg*;

This predicate procedure is called each time an event is added to the event queue. After the predicate procedure finds a match, **XPeekIfEvent** copies the matched event into the client-supplied **XEvent** structure without removing the event from the queue. If it blocks waiting for an event, **XPeekIfEvent** flushes the output buffer.

¦ Copyright IBM Corp. 1985, 1991

2.4.236 XPending

int XPending(*display*)
Display **display*;

display Specifies the connection to the X Server.

XPending returns the number of events received from the X Server but not yet removed from the event queue. Use **XNextEvent** or **XWindowEvent** to remove the events from the queue.

XPending is identical to **XEventsQueued** with the **QueuedAfterFlush** as the mode attribute.

¦ Copyright IBM Corp. 1985, 1991

2.4.237 Xpermalloc

char *Xpermalloc(*size*)
unsigned int *size*;

size Amount of memory in bytes to allocate.

Xpermalloc allocates some memory that will never be returned. This routine is used by toolkits to allocate permanent storage when increased performance and space savings are wanted.

¦ Copyright IBM Corp. 1985, 1991

2.4.238 XPointInRegion

int XPointInRegion(*region, x, y*)
Region *region*;
int *x, y*;

region Specifies the region.

x, y Specifies the *x* and *y* coordinates of the point.

XPointInRegion determines if a specified point resides in a specified region. It returns a non-zero if the point, which is defined by the *x* and *y* coordinates, is contained in the specified region.

¦ Copyright IBM Corp. 1985, 1991

2.4.239 XPolygonRegion

Region XPolygonRegion (*points, n, fill_rule*)
XPoint *points[]*;
int *n*;
int *fill_rule*;

points Specifies an array of points.

n Specifies the number of points in the polygon.

fill_rule Specifies the fill rule (**EvenOddRule** or **WindingRule**) to be set for the specified graphics context.

XPolygonRegion generates a region from points. It returns a region defined by the points array. (See "Drawing Points, Lines, Rectangles, and Arcs" in topic 1.8.3.)

¦ Copyright IBM Corp. 1985, 1991

2.4.240 XPutBackEvent

XPutBackEvent(*display, event*)
Display **display*;
XEvent **event*;

display Specifies the connection to the X Server.

event Specifies a pointer to the **XEvent** structure.

XPutBackEvent pushes an event back to the top of the event queue of the current display. There is no limit to the number of times you can do this.

¦ Copyright IBM Corp. 1985, 1991

2.4.241 XPutImage

XPutImage(*display, drawable, gc, image, src_x, src_y, dst_x, dst_y, width, hei*
Display **display*;
Drawable *drawable*;
GC *gc*;
XImage **image*;
int *src_x, src_y*;
int *dst_x, dst_y*;
unsigned int *width, height*;

display Specifies the connection to the X Server.

drawable Specifies the drawable.

gc Specifies the graphics context.

image Specifies the image to be combined with the rectangle.

src_x Specifies the offset in X from the left edge of the image defined by the **XImage** data structure.

src_y Specifies the offset in Y from the top edge of the image defined by the **XImage** data structure.

dst_x, dst_y Specifies the *x* and *y* coordinates for the subimage. These coordinates, which are relative to the origin of the drawable, define where the image will be drawn.

width, height Specifies the *width* and *height* which define the dimensions of the rectangle of the subimage.

XPutImage combines an image in memory with a rectangle of the drawable on your display. (See "Transferring Images Between Client and Server" in topic 1.9.3 for information about images.)

If **XYBitmap** format is used, the depth must be one, and the image must be **XYFormat**. The foreground pixel in the **GC** defines the source for the one bits in the image. The background pixel defines the source for the zero bits.

For **XYPixmap** and **ZPixmap**, the depth must match the depth of drawable.

For **XYPixmap**, the image must be sent in **XYFormat**.

For **ZPixmap**, the image must be sent in the **ZFormat** defined for the given depth. The section of the image defined by *src_x*, *src_y*, *width*, and *height* are drawn on the specified part of the drawable.

XPutImage uses the graphics context components: *function, plane_mask, subwindow_mode, clip_x_origin, clip_y_origin*, and *clip_mask*. It also uses the graphics context mode-dependent components: *foreground* and *background*. See "Manipulating Graphics Context or State" in topic 1.7.8.

XPutImage can generate the event errors **BadDrawable**, **BadGC**, **BadMatch**, and **BadValue**.

¦ Copyright IBM Corp. 1985, 1991

2.4.242 XPutPixel

int XPutPixel(*ximage, x, y, pixel*)
XImage **ximage*;
int *x*;
int *y*;
unsigned long *pixel*;

ximage Specifies a pointer to the image.

x, y Specifies the *x*, *y* coordinates which are relative to the origin of the image.

pixel Specifies the new pixel value.

XPutPixel sets a pixel value in an image. It overwrites the pixel in the specified image with the specified pixel value. (See "Transferring Images Between Client and Server" in topic 1.9.3 for information about images.) The input pixel value must be in normalized format. The least-significant byte of the long is the least-significant byte of the pixel.

¦ Copyright IBM Corp. 1985, 1991

2.4.243 XQueryBestCursor

Status XQueryBestCursor(*display, drawable, width, height, width_return, height*
Display **display*;
Drawable *drawable*;
unsigned int *width, height*;
unsigned int **width_return, *height_return*;

display	Specifies the connection to the X Server.
drawable	Specifies the drawable of the desired screen.
width, height	Specifies the *width* and *height* of the cursor.
width_return, height_return	Returns the width and height dimensions which are closest to the specified width and height.

XQueryBestCursor provides a way to determine the best size cursors for the display. It returns the largest size that can be displayed. It returns a bitmap shape for the cursor shape which is acceptable for **XCreatePixmapCursor**.

XQueryBestCursor can generate the error **BadDrawable**.

¦ Copyright IBM Corp. 1985, 1991

2.4.244 XQueryBestSize

Status XQueryBestSize(*display, class, drawable, width, height, width_return, height_return*)
Display **display*;
int *class*;
Drawable *drawable*;
unsigned int *width, height*;
unsigned int **width_return, *height_return*;

display	Specifies the connection to the X Server.
class	Specifies the class.
drawable	Specifies the drawable.
width, height	Specifies the width and height.
width_return, height_return	Returns the width and height of the drawable best supported by the display hardware.

XQueryBestSize obtains the best size of a tile, stipple, or cursor. The *drawable* indicates the screen, and possibly, the class and the depth of the window. The *class* variable can be set to one of the following:

CursorShape, which is the largest size that can be displayed on the specified screen.

TileShape, which is the size that can be tiled fastest on the specified screen.

StippleShape, which is the size that can be stippled fastest on the specified screen.

An **InputOnly** window cannot be used as the drawable for **TileShape** or **StippleShape**.

See "Manipulating Graphics Context or State" in topic 1.7.8 for more information on graphics.

XQueryBestSize can generate the errors **BadDrawable**, **BadMatch**, and **BadValue**.

¦ Copyright IBM Corp. 1985, 1991

2.4.245 XQueryBestStipple

Status XQueryBestStipple(*display, drawable, width, height, width_return, heig*
Display **display*;
Drawable *drawable*;
unsigned int *width, height*;
unsigned int **width_return, *height_return*;

display	Specifies the connection to the X Server.
drawable	Specifies any drawable on a screen.
width, height	Specifies the *width* and *height*.
width_return, height_return	Returns the width and height of the drawable best supported by the display hardware.

XQueryBestStipple obtains the best stipple shape that can be stippled fastest on the screen. The *drawable* indicates the screen and, possibly, the window class and depth. An **InputOnly** window cannot be used as the drawable for this function.

XQueryBestStipple can generate the event errors **BadDrawable** and **BadMatch**.

¦ Copyright IBM Corp. 1985, 1991

2.4.246 XQueryBestTile

Status XQueryBestTile(*display, drawable, width, height, width_return, height_r*
Display **display*;
Drawable *drawable*;
unsigned int *width*, *height*;
unsigned int **width_return, *height_return*;

display	Specifies the connection to the X Server.
drawable	Specifies any drawable on a screen.
width, height	Specifies the *width* and *height*.
width_return, height_return	Returns the width and height of the drawable best supported by the display hardware.

XQueryBestTile obtains the best fill pattern that can be tiled fastest on the specified screen. The drawable indicates the screen and, possibly, the class and depth of the window. An **InputOnly** window cannot be used as the drawable for this function.

XQueryBestTile can generate the event errors **BadDrawable** and **BadMatch**.

¦ Copyright IBM Corp. 1985, 1991

2.4.247 XQueryColor

XQueryColor(*display, cmap, def_return*)
Display **display*;
Colormap *cmap*;
XColor **def_return*;

display	Specifies the connection to the X Server.
cmap	Specifies the colormap ID.
def_return	Returns the RGB values for the pixel specified in the structure.

XQueryColor obtains the color values for a single specified pixel value. It returns the red, green, and blue color values stored in *cmap* for the pixel value passed to the pixel member of the **XColor** structure. **XQueryColor** sets the *flag* member in the **XColor** structure to all three colors. The returned values for an unallocated entry are undefined. See "Creating Colormaps" in topic 1.7.1.

XQueryColor can generate the event errors **BadColor** and **BadValue**.

¦ Copyright IBM Corp. 1985, 1991

2.4.248 XQueryColors

XQueryColors(*display, cmap, defs_return, ncolors*)
Display **display*;
Colormap *cmap*;
XColor *defs_return*[];
int *ncolors*;

display	Specifies the connection to the X Server.
cmap	Specifies the colormap ID.
defs_return	Returns an array of color definition structures.
ncolors	Specifies the number of **XColor** structures in the color definition array.

XQueryColors obtains color values for an array of pixels stored in color structures. It returns the red, green, and blue color values stored in *cmap* for the pixel value passed to an array of pixel members of **XColor** structures. **XQueryColors** sets the *flag* member in the **XColor** structure to all three colors. See "Creating Colormaps" in topic 1.7.1.

XQueryColors can generate the event errors **BadColor** and **BadValue**.

¦ Copyright IBM Corp. 1985, 1991

2.4.249 XQueryFont

XFontStruct *__**XQueryFont**__(*display, font_ID*)
Display **display*;
XID *font_ID*;

display Specifies the connection to the X Server.

font_ID Specifies the font ID or the graphics context whose current information is wanted.

XQueryFont returns information about a loaded font. It returns a pointer to the **XFontStruct** structure. It queries a font or the fonts stored in the GC. If this query is not successful, it returns **NULL**.

If the font is stored in the GC, the font ID in **XFontstruct** is the ID of the GC. Be careful when using this ID in other functions. For example, the ID of the **GC** is not valid as a font ID in a **Set** or **Get** font function.

Use **XFreeFontInfo** to free this data.

¦ Copyright IBM Corp. 1985, 1991

2.4.250 XQueryKeymap

XQueryKeymap(*display, keys_return*)
Display **display*;
char *keys_return[32]*;

display Specifies the connection to the X Server.

keys_return Returns an array of bytes that identifies which keys are pressed down.

XQueryKeymap returns a bit vector for the keyboard, where each one bit indicates that the corresponding key is currently pressed down. The vector is represented as 32 bytes. Byte **N** from 0 contains the bits for keys **8N** to **8N+7** with the least significant bit in the byte representing key **8N**.

¦ Copyright IBM Corp. 1985, 1991

2.4.251 XQueryPointer

Bool XQueryPointer(*display, window, root_return, child_return, root_x_return,*
root_y_return, win_x_return, win_y_return, mask_re
Display **display*;
Window *window;
Window **root_return, *child_return*;
int **root_x_return, *root_y_return*;
int **win_x_return, *win_y_return*;
unsigned int **mask_return*;

display Specifies the connection to the X Server.

window Specifies the window ID.

root_return Returns the root window ID for the window on which the pointer is currently located.

child_return Returns the child window ID that the pointer is located in, if any.

root_x_return Returns the pointer *x* coordinate which is relative to the origin of the root window.

root_y_return Returns the pointer *y* coordinate which is relative to the origin of the root window.

win_x_return Returns the pointer *x* coordinate which is relative to the specified window.

win_y_return Returns the pointer *y* coordinate which is relative to the specified window.

mask_return Returns the current state of the modifier keys and pointer buttons.

XQueryPointer returns the root window the pointer is logically on and the pointer coordinates relative to the origin of the root window. (3)

If **XQueryPointer** returns **True**, the pointer coordinates returned to *win_x_return* and *win_y_return* are relative to the origin of the specified window, and it returns the child ID of the window containing the pointer, if any.

If **XQueryPointer** returns **False**, the pointer is not on the same screen as the specified window. In this case, **XQueryPointer** returns **None** to *child_return* and zero to *win_x_return* and *win_y_return*.

The current logical state of the keyboard buttons and the modifier keys are returned in the *mask_return*. Depending on the current state of the buttons and the modifier keys, **XQueryPointer** can set this argument to the bitwise inclusive OR of one or more of the button or modifier key bitmasks.

Note that the logical state of a device as seen by the X protocol may lag the physical state if the device event processing is frozen. See "Grabbing the Pointer" in topic 1.10.2.

XQueryPointer can generate the event error **BadWindow**.

¦ Copyright IBM Corp. 1985, 1991

(3) AIX supports only one screen.

¦ Copyright IBM Corp. 1985, 1991

2.4.252 XQueryTextExtents

XQueryTextExtents(*display, font_ID, string, nchars, direction_return, ascent_return, descent_return, overall_return*)
Display **display*;
XID *font_ID*;
char **string*;
int *nchars*;
int **direction_return*;
int **ascent_return, *descent_return*;
XCharStruct **overall_return*;

display Specifies the connection to the X Server.

font_ID Specifies either the font ID or the graphics context that contains the font.

string Specifies the character string.

nchars Specifies the number of characters in the character string.

direction_return Returns the value of the direction (**FontLeftToRight** or **FontRightToLeft**) hint member.

ascent_return Returns the font ascent member which is the maximum of the ascent metrics of all characters in the string.

descent_return Returns the font descent member which is the maximum of the descent metrics.

overall_return Returns the overall size in the specified **XCharStruct** structure.

XQueryTextExtents queries the server for the sizes of an 8-bit character string. It returns the logical extents of the specified 8-bit character string in the specified font or the font contained in the specified **GC**.

XQueryTextExtents returns a **XCharStruct** structure whose members are set to the values specified. The *width* member is set to the sum of the character-width metrics of all characters in the string. For each character in the string:

W should be the sum of the character-width metrics of all characters preceding it in the string.
L should be the left-side-bearing metric of the character plus *W*.
R should be the right-side-bearing metric of the character plus *W*.
lbearing member is set to the minimum *L* of all characters in the string.
rbearing member is the maximum *R*.

If the font has no defined default character, undefined characters in the string are zero. See "Manipulating Fonts" in topic 1.9.

XQueryTextExtents can generate the event errors **BadFont** and **BadGC**.

¦ Copyright IBM Corp. 1985, 1991

2.4.253 XQueryTextExtents16

XQueryTextExtents16(*display, font_ID, string, nchars, direction_return, ascent_return, descent_return, overall_return*)
Display **display*;
XID *font_ID*;
XChar2b **string*;
int *nchars*;
int **direction_return*;
int **ascent_return, *descent_return*;
XCharStruct **overall_return*;

display Specifies the connection to the X Server.

font_ID Specifies the font ID or the graphics context that contains the font.

string Specifies the character string.

nchars Specifies the number of characters in the character string.

direction_return Returns the value of the direction (**FontLeftToRight** or **FontRightToLeft**) hint member.

ascent_return Returns the font ascent member which is the maximum of the ascent metrics of all characters in the string.

descent_return Returns the font descent member which is the maximum of the descent metrics.

overall_return Returns the overall size in the specified **XCharStruct** structure.

XQueryTextExtents16 queries the server for the sizes of a 16-bit character string. It returns the logical extents of the specified character string in the specified font or the font contained in the specified **GC**.

XTextExtents16 returns a **XCharStruct** structure whose members are set to the specified values. The *width* member is set to the sum of the character-width metrics of all characters in the string. For each character in the string:

W should be the sum of the character-width metrics of all characters preceding it in the string.
L should be the left-side-bearing metric of the character plus *W*.
R should be the right-side-bearing metric of the character plus *W*.
lbearing member is set to the minimum *L* of all characters in the string.
rbearing member is the maximum *R*.

If the font has no defined default character, the undefined characters in the string are zero. See "Manipulating Fonts" in topic 1.9.

XQueryTextExtents16 can generate the event errors **BadFont** and **BadGC**.

¦ Copyright IBM Corp. 1985, 1991

2.4.254 XQueryTree

Status XQueryTree(*display, window, root_return, parent_return, children_return, nchildren_return*)
Display **display*;
Window *window*;
Window **root_return*;
Window **parent_return*;
Window ***children_return*;
unsigned int **nchildren_return*;

display	Specifies the connection to the X Server.
window	Specifies the window ID.
root_return	Returns the root window ID for the specified window.
parent_return	Returns the parent window ID for the specified window.
children_return	Returns a pointer to the list of children for the specified window.
nchildren_return	Returns the number of children for the specified window.

XQueryTree returns the root ID, the parent window ID, a pointer to the list of children windows, and the number of children in the list for the specified window. The children are listed in current stacking order from bottom-most (first) to top-most (last).

If **XQueryTree** fails, it returns a zero. If it succeeds, it returns a non-zero.

Use **XFree** to free this list when it is no longer needed.

XQueryTree can generate the event error **BadWindow**.

¦ Copyright IBM Corp. 1985, 1991

2.4.255 XRaiseWindow

XRaiseWindow(*display, window*)
Display **display*;
Window *window*;

display Specifies the connection to the X Server.

window Specifies the window ID.

XRaiseWindow raises the specified window to the top of the stack so that no sibling window obscures it. If the windows are regarded as overlapping sheets of paper stacked on a desk, then raising a window is analogous to moving the sheet to the top of the stack but leaving its *x* and *y* location on the desk constant.

Raising a mapped window may generate exposure events for the window and any mapped subwindows that were formerly obscured. See "Defining Window Attributes" in topic 1.5.3 for information on window attributes that can affect this routine.

XRaiseWindow can generate the event error **BadWindow**.

¦ Copyright IBM Corp. 1985, 1991

2.4.256 XReadBitmapFile

int XReadBitmapFile(*display, drawable, filename, width_return, height_return, x_hot_return, y_hot_return*)
Display **display*;
Drawable *drawable*;
char **filename*;
int **width_return*, **height_return*;
Pixmap **bitmap_return*;
int **x_hot_return*, **y_hot_return*;

display Specifies the connection to the X Server.

drawable Specifies the drawable.

filename Specifies the filename to use. The format of the filename depends on the operating system.

width_return Returns the width value of the read in bitmap file.

height_return Returns the height values of the read in bitmap file.

bitmap_return Returns the bitmap ID created.

x_hot_return Returns the *x* hot spot coordinate.

y_hot_return Returns the *y* hot spot coordinates.

XReadBitmapFile reads a file containing a bitmap. This file can be in the standard X-Windows 1.1 format--the format used by X-Windows 1.1 bitmap program--or in the X-Windows 2.1 bitmap format.

XReadBitmapFile assigns the height and width from the bitmap file that was read to the width and height of the target bitmap file or the file initiating the call. It then creates a pixmap, reads the bitmap data from the file into the pixmap and assigns the pixmap to the bitmap of the target file.

When this function is completed, free the bitmap with **XFreePixmap**.

If *x_hot* and *y_hot* have assigned values, **XReadBitmapFile** returns these values to the bitmap file. If a hotspot is not defined, **XReadBitmapFile** sets **x_hot* and **y_hot* to **-1, -1**.

XReadBitmapFile returns one of the following:

BitmapOpenFailed if the file cannot be opened.

BitmapFileInvalid if the file can be opened but it contains invalid bitmap data.

BitmapNoMemory if insufficient working space was allocated.

BitmapSuccess if the file is readable and valid.

¦ Copyright IBM Corp. 1985, 1991

2.4.257 XRebindCode

XRebindCode (*display, keycode, shiftbits, str, nbytes*)
Display **display*;
unsigned int *keycode*;
unsigned int *shiftbits*;
char ***string*;
int *nbytes*;

display Specifies the connection to the X Server.

keycode Specifies keycode to change temporarily.

shiftbits Specifies shift bits.

string Returns a pointer to the string.

nbytes Specifies the number of bytes in the string.

XRebindCode rebinds the keyboard temporarily. It changes the binding of the keyboard. (See **XLookupMapping** in this chapter.) After issuing **XRebindCode**, subsequent calls to **XLookupMapping** returns the supplied string instead of the string found in the keymap file. The string should be stored in static storage; an automatic string may be deallocated by the time it is needed.

If *nbytes* is zero and *string* is not **NULL**, then *string* points to a 2-byte array that contains the code page and code point of a dead key. If *string* is **NULL** and *nbytes* is not zero, then *nbytes* defines a function ID.

See *AIX X-Windows User's Guide* for information about the **keycomp** command. See *AIX Operating System Technical Reference* for more information on function IDs, code pages and code points.

¦ Copyright IBM Corp. 1985, 1991

2.4.258 XRebindKeysym

XRebindKeysym(*display, keysym, list, mod_count, string, bytes_string*)
Display **display*;
KeySym *keysym*;
KeySym **list*;
int *mod_count*;
unsigned char **string*;
int *bytes_string*;

display Specifies the connection to the X Server.

keysym Specifies the keysym to be rebound.

list Specifies a pointer to an array of keysyms that are being used as modifiers.

mod_count Specifies the number of modifiers in the modifier list.

string Specifies a pointer to the string to be returned by **XLookupString**.

bytes_string Specifies the length of the string.

XRebindKeysym rebinds the meaning of a keysym for a client. It does not redefine the keycode in the X Server but provides a way to attach long strings to keys. **XLookupString** returns this string when the appropriate set of modifier keys are pressed and when the keysym is used for the translation. You can rebind a keysym that may not exist.

¦ Copyright IBM Corp. 1985, 1991

2.4.259 XRecolorCursor

XRecolorCursor(*display, cursor, foreground_color, background_color*)
Display **display*;
Cursor *cursor*;
XColor **foreground_color, *background_color*;

display Specifies the connection to the X Server.

cursor Specifies the cursor.

foreground_color Specifies the red, green, and blue (RGB) values for the foreground of the source.

background_color Specifies the red, green, and blue (RGB) values for the background of the source.

XRecolorCursor changes the color of the specified cursor. If the cursor is being displayed on a screen, this change is visible immediately. For information about color, see "Creating Colormaps" in topic 1.7.1.

XRecolorCursor can generate the error **BadCursor**.

¦ Copyright IBM Corp. 1985, 1991

2.4.260 XRectInRegion

int XRectInRegion(*region, x, y, width, height*)
Region *region*;
int *x, y*;
unsigned int *width, height*;

region	Specifies the region.
x, y	Specifies the *x* and *y* coordinates which define the location of the point.
width, height	Specifies the *width* and *height* of the rectangle in which the point may reside.

XRectInRegion determines if a specified rectangle resides in the specified region. **XRectInRegion** returns one of the following:

RectangleIn if the rectangle is entirely in the region specified.

RectangleOut if the rectangle is entirely out of the region specified.

RectanglePart if the rectangle is partially in the specified region.

¦ Copyright IBM Corp. 1985, 1991

2.4.261 XRefreshKeyboardMapping

XRefreshKeyboardMapping(*event_map*)
XMappingEvent **event_map*;

event_map Specifies the mapping event to be used.

XRefreshKeyboardMapping refreshes the stored modifier and keymap information. Usually, this function is called when a **MappingNotify** event occurs to update a
client's knowledge of the keyboard. See page "Processing MappingNotify Events" in topic 1.11.15.7.

¦ Copyright IBM Corp. 1985, 1991

2.4.262 XRemoveFromSaveSet

XRemoveFromSaveSet(*display, window_remove*)
Display **display*;
Window *window_remove*;

display Specifies the connection to the X Server.

window_remove Specifies the window ID of the window to be removed.

XRemoveFromSaveSet removes the specified window and the children of the specified window from the client's save-set. The specified window must be created by another client. The X Server automatically removes windows from the save-set when the windows are destroyed.

XRemoveFromSaveSet can generate the event errors **BadMatch** and **BadWindow**.

¦ Copyright IBM Corp. 1985, 1991

2.4.263 XRemoveHost

XRemoveHost(*display, host*)
Display **display*;
XHostAddress **host*;

display Specifies the connection to the X Server.

host Specifies the network address of the host system.

XRemoveHost removes the specified host from the access control list for that display. The display must be on the same host as the client process. (See "Controlling Host Access" in topic 1.10.6.)

If you remove your system from the access list, you will not be able to connect to that server. To regain access to the server, you must reset the server.

XRemoveHost can generate the event errors **BadAlloc** and **BadValue**.

¦ Copyright IBM Corp. 1985, 1991

2.4.264 XRemoveHosts

XRemoveHosts(*display, hosts, num_hosts*)
Display **display*;
XHostAddress **hosts*;
int *num_hosts*;

display Specifies the connection to the X Server.

hosts Specifies each host to be removed.

num_hosts Specifies the number of hosts.

XRemoveHosts removes each specified host from the access control list for that display. The display must be on the same host as the client process. (See "Controlling Host Access" in topic 1.10.6.)

If you remove your system from the access list, you will not be able to connect to that server. To regain access to the server, you must reset the server.

XRemoveHosts can generate the event errors **BadAlloc** and **BadValue**.

¦ Copyright IBM Corp. 1985, 1991

2.4.265 XReparentWindow

XReparentWindow(*display, window, parent, x, y*)
Display **display*;
Window *window*;
Window *parent*;
int *x, y*;

display Specifies the connection to the X Server.

window Specifies the window ID.

parent Specifies the parent window ID.

x, y Specifies the *x* and *y* coordinates which define the position of the specified window in the new parent window.

XReparentWindow reparents the specified window by inserting it as the child of the specified parent. If the specified window is mapped, **XReparentWindow** automatically performs an **XUnmapWindow** request on it. Then, **XReparentWindow** removes the specified window from its current position in the hierarchy and inserts it as the child of the specified parent. The window is placed on top in the stacking order with respect to the sibling windows.

After reparenting the specified window, the X Server generates a **ReparentNotify** event. (See "Processing ReparentNotify Events" in topic 1.11.15.8.)

The *override_redirect* member of the structure returned by this event can be set to **True** or **False**. If **True**, clients normally ignore this event.

Finally, if the specified window was mapped originally, **XReparentWindow** performs a **XMapWindow** request on it automatically.

The X Server performs normal exposure processing on formerly obscured windows. The X Server might not generate exposure events for regions from the initial **XUnmapWindow** request that are immediately obscured by the final **XMapWindow** request.

A **BadMatch** error is generated if one of the following occurs:

The new parent window is not on the same screen as the old paren window.

The new parent window is the specified window or an inferior of th specified window.

The specified window has a **ParentRelative** background and the new parent window is not the same depth as the specified window.

XReparentWindow also can generate the event error **BadWindow**.

¦ Copyright IBM Corp. 1985, 1991

2.4.266 XResetScreenSaver

XResetScreenSaver(*display*)
Display **display*;

display Specifies the connection to the X Server.

XResetScreenSaver resets the screensaver. (See **XSetScreenSaver** in this chapter.)

¦ Copyright IBM Corp. 1985, 1991

2.4.267 XResizeWindow

XResizeWindow(*display, window, width, height*)
Display **display*;
Window *window*;
unsigned int *width, height*;

display Specifies the connection to the X Server.

window Specifies the window ID.

width, height Specifies the *width* and *height* of the window after this function is completed.

XResizeWindow changes the inside dimensions of the specified window. It does not change the borders, the upper-left coordinates, or the origin of the window. It does not raise the window. A mapped window may or may not lose its contents after it is resized. A mapped window generates an **Expose** event if it loses its contents. If a mapped window is made smaller, exposure events are generated on windows that it formerly obscured. (See "Configuring Windows" in topic 1.5.7.)

If the *override_redirect* attribute of the window is **False** and another client has selected **SubstructureRedirectMask** on the parent window, a **ConfigureRequest** event is generated, and no further processing is performed. (See "Processing ConfigureNotify Events" in topic 1.11.15.2.)

XResizeWindow can generate the event error **BadWindow**.

¦ Copyright IBM Corp. 1985, 1991

2.4.268 XResourceManagerString

char *XResourceManagerString(*display*)
Display **display*;

display Specifies the connection to the X Server.

XResourceManagerString returns the RESOURCE_MANAGER property from the screen of the root window of screen zero. This is the value returned by **XOpenDisplay**.

¦ Copyright IBM Corp. 1985, 1991

2.4.269 XRestackWindows

XRestackWindows(*display, windows, nwindows*)
Display * *display*;
Window *windows[]*;
int *nwindows*;

display Specifies the connection to the X Server.

windows Specifies an array containing the windows to be restacked. The specified windows must have the same parent.

nwindows Specifies the number of windows to be restacked.

XRestackWindows restacks the windows in the order (from top to bottom) specified. The stacking order of the first window in the windows array is unaffected, but the other windows in the array are stacked underneath the first window in the order of the array. The stacking order of the other windows is not affected. (See "Configuring Windows" in topic 1.5.7.)

If the *override_redirect* attribute of the window is **False** and another client has selected **SubstructureRedirectMask** on the parent window, a **ConfigureRequest** event is generated for each window whose *override_redirect* is not set, and no further processing is performed. Otherwise, the windows will be restacked in top-to-bottom order. (See "Processing ConfigureNotify Events" in topic 1.11.15.2.)

XRestackWindows can generate the event error **BadWindow**.

¦ Copyright IBM Corp. 1985, 1991

2.4.270 XrmGetFileDatabase

XrmDatabase XrmGetFileDatabase(*filename*)
char **filename*;

filename Specifies the resource database filename.

XrmGetFileDatabase retrieves a database from nonvolatile storage. It opens the specified file, creates a new resource database, and loads it with the specifications read in from the specified file. The specified file must contain lines in the format accepted by **XrmPutLineResource**. If **XrmGetFileDatabase** cannot open the specified file, it returns **NULL**.

¦ Copyright IBM Corp. 1985, 1991

2.4.271 XrmGetResource

Bool XrmGetResource(*database, str_name, str_class, str_type_return, str_value_*
XrmDatabase *database*;
char **str_name*;
char **str_class*;
char ***str_type_return*;
XrmValue **str_value_return*;

database	Specifies the database to be used.
str_name	Specifies the fully qualified name (as a string) of the value being retrieved.
str_class	Specifies the fully qualified class (as a string) of the value being retrieved.
str_type_return	Returns a pointer (as a string) to the representation type of the destination.
str_value_return	Returns the value in the database.

XrmGetResource retrieves a resource from a resource database.

¦ Copyright IBM Corp. 1985, 1991

2.4.272 XrmGetStringDatabase

XrmDatabase XrmGetStringDatabase(*data*)
char **data*;

data Specifies the database contents using a string.

XrmGetStringDatabase creates a new database and stores the resources specified in the specified null-terminated string. It reads the information out of a string. Each line is separated by a new line character in the format accepted by **XrmPutLineResource**.

¦ Copyright IBM Corp. 1985, 1991

2.4.273 XrmInitialize

void XrmInitialize()

XrmInititalize initializes the resource manager.

¦ Copyright IBM Corp. 1985, 1991

2.4.274 XrmMergeDatabases

void XrmMergeDatabases(*source_db, target_db*)
XrmDatabase *source_db, *target_db*;

source_db Specifies the resource database to be merged in the target database.

target_db Specifies a pointer to the resource database where the source database is to be merged.

XrmMergeDatabases merges the contents of one database into another. It may overwrite entries in the destination database. It is used to combine databases such as an application specific database of defaults and a database of user preferences. The original database is destroyed.

¦ Copyright IBM Corp. 1985, 1991

2.4.275 XrmParseCommand

void XrmParseCommand(*db, table, table_count, name, argc_return, argv_return*,)
XrmDatabase **db*;
XrmOptionDescList *table*;
int *table_count*;
char **name*;
int **argc_return*;
char ***argv_return*;

db Specifies a pointer to the resource database.

table Specifies table of command line arguments to be parsed.

table_count Specifies the number of entries in the table.

name Specifies the application name.

argc_return Contains the number of arguments before the call. Returns the number of remaining arguments after the call.

argv_return Returns a pointer to the command line arguments before the call. Returns the matched arguments that were removed after the call.

XrmParseCommand loads a database from a C language command line according to the following:

It parses an *argc, argv* pair according to the specified option table
It loads recognized options into the specified databas
It modifies the *argc, argv* pair to remove all recognized options.

If the resource database contains **NULL**, a new resource database is created and a pointer is returned to the new resource database in database.

The specified table is used to parse the command line. Recognized entries in the table are removed from *argv_return*, and entries are made in the specified resource database. The table entries contain information on the option string, the option name, the style of option and a value to provide if the option kind is **XrmOptionNoArg**.

Use the application name as the *name* argument. This argument is prepended to the *resourceName* in the option table before storing the specification. (See "Using the Resource Manager" in topic 1.13 for a definition of the **XrmOptionDescRec** structure.)

The *argc_return* argument specifies the number of arguments in *argv_return* and is set to the remaining number of arguments that were not parsed.

¦ Copyright IBM Corp. 1985, 1991

2.4.276 XrmPutFileDatabase

void XrmPutFileDatabase(*database, stored_db*)
XrmDatabase *database*;
char **stored_db*;

database Specifies the database to be used.

stored_db Specifies the filename for the stored database.

XrmPutFileDatabase stores a copy of the current database of the application in the specified file. The file is an ASCII text file that contains lines in the format accepted by **XrmPutLineResource**.

¦ Copyright IBM Corp. 1985, 1991

2.4.277 XrmPutLineResource

void XrmPutLineResource(*database, line*)
XrmDatabase **database*;
char **line*;

database Specifies a pointer to the resource database.

line Specifies the resource value pair as a single string. A colon (":") separates the name from the value.

XrmPutLineResource adds a single resource entry to the specified database. The resource entry is specified as a string that contains both a name and a value pair. Any space before or after the name or colon in the *line* argument is ignored. The value is terminated by a new-line or a **NULL** character.

The value pair may contain embedded new-line characters prefixed by the "\" and "n" character pair, which are removed before the value is stored in the database. For example, *line* might have the value

aixterm*background:green

Null-terminated strings without a new line are also permitted.

If *database* is **NULL**, a new resource database is created and a pointer to the new resource database is returned in the *database* argument.

¦ Copyright IBM Corp. 1985, 1991

2.4.278 XrmPutResource

void XrmPutResource(*database, specifier, type, value*)
XrmDatabase **database*;
char **specifier*;
char **type*;
XrmValue **value*;

database Specifies a pointer to the resource database.

specifier Specifies a partial specification of the resource.

type Specifies the type of the resource.

value Specifies the value of the resource.

XrmPutResource calls **XrmStringToBindingQuarkList** followed by this expression:

```
XrmQPutResource(database, bindings, quarks,
                XrmStringToQuarkList(type), value)
```

XrmPutResource stores resources into the database. This functions takes a partial resource specification, a representation type, and a value. This value is copied into the specified database.

If the *database* argument is **NULL**, a new resource database is created and a pointer to the new resource database is returned in this argument.

¦ Copyright IBM Corp. 1985, 1991

2.4.279 XrmPutStringResource

void XrmPutStringResource(*database, resource, value*)
XrmDatabase **database*;
char **resource*;
char **value*;

database	Specifies a pointer to the resource database.
resource	Specifies the resource as a string.
value	Specifies the value of the resource which is specified as a string.

XrmPutStringResource adds a resource with the specified value to the specified database. It takes both the resource and value as strings, converts them to quarks, and then calls **XrmQPutResource**.

If the *database* argument is **NULL**, a new resource database is created and a pointer to the new database is returned in this argument.

¦ Copyright IBM Corp. 1985, 1991

2.4.280 XrmQGetResource

Bool XrmQGetResource(*database, quark_name, quark_class, quark_type_return, value_return*)
XrmDatabase *database*;
XrmNameList *quark_name*;
XrmClassList *quark_class*;
XrmRepresentation **quark_type_return*;
XrmValue **value_return*;

database	Specifies the database to be used.
quark_name	Specifies the fully qualified name (as a quark) of the value being retrieved.
quark_class	Specifies the fully qualified class (as a quark) of the value being retrieved.
quark_type_return	Returns a pointer (as a quark) to the representation type of the destination.
value_return	Returns the value in the database.

XrmQGetResource gets a resource from a resource database.

¦ Copyright IBM Corp. 1985, 1991

2.4.281 XrmQGetSearchList

Bool XrmQGetSearchList(*database, names, classes, list_return, list_length*)
XrmDatabase *database*;
XrmNameList *names*;
XrmClassList *classes*;
XrmSearchList *list_return*;
int *list_length*;

database Specifies the database to be used.

names Specifies a list of resource names.

classes Specifies a list of resource classes.

list_return Returns a search list for further use.

list_length Specifies the number of entries (not the byte size) allocated for *list_return*.

XrmQGetSearchList takes a list of names and classes and returns a list of database levels where a match might occur. (You must allocate sufficient space for the list before using this function.) The returned list is in a best-to-worst order. It uses the same algorithm as **XrmGetResource** for determining precedence.

XrmQGetSearchList returns **True** if *list_return* is large enough for the search list. It returns **False** if *list_return* is not large enough.

The size of the search list is dependent upon the number of levels and wildcards in the resource specifiers stored in the database. The worst case length is 3(*n*), where *n* is the number of name or class components in *names* or *classes*.

When using **XrmQGetSearchList** followed by multiple probes for resources with a common name and class prefix, only the common prefix should be specified in the name and class list to **XrmQGetSearchList**.

¦ Copyright IBM Corp. 1985, 1991

2.4.282 XrmQGetSearchResource

Bool XrmQGetSearchResource(*list, name, class, type_return, value_return*)
XrmSearchList *list*;
XrmName *name*;
XrmClass *class*;
XrmRepresentation **type_return*;
XrmValue **value_return*;

list Specifies the search list returned by **XrmWGetSearchList**.

name Specifies the resource name.

class Specifies the resource class.

type_return Returns data representation type.

value_return Returns the value in the database.

XrmQGetSearchResource searches the specified database levels for the resource identified by the specified name and class. The search stops with the first match. If the resource is found, this routine returns **True**.

A call to **XrmQGetSearchList** with a name and class list containing all but the last component of a resource name followed by **XrmQGetSearchResource** with the last component name and class, returns the same database entry as **XrmGetResource** and **XrmQGetResource** with the fully qualified name and class.

¦ Copyright IBM Corp. 1985, 1991

2.4.283 XrmQPutResource

void XrmQPutResource(*database, bindings, quarks, type, value*)
XrmDatabase **database*;
XrmBindingList *bindings*;
XrmQuarkList *quarks*;
XrmRepresentation *type*;
XrmValue **value*;

database Specifies a pointer to the database.

bindings Specifies a list of bindings.

quarks Specifies the partial name or class list of the resource to be stored.

type Specifies the type of the resource.

value Specifies the value of the resource.

XrmQPutResource stores resources into the database. It takes a partial resource specification, a representation type, and a value. This value is copied into the specified database.

If the *database* argument contains **NULL**, a new resource database is created and a pointer to the new database is returned in this argument.

¦ Copyright IBM Corp. 1985, 1991

2.4.284 XrmQPutStringResource

void XrmQPutStringResource(*database, bindings, quarks, value*)
XrmDatabase **database*;
XrmBindingList *bindings*;
XrmQuarkList *quarks*;
char **value*;

database Specifies a pointer to the resource database.

bindings Specifies a list of bindings.

quarks Specifies the partial name or class list of the resource to be stored.

value Specifies the value (as a string) of the resource.

XrmQPutStringResource adds a string resource using quarks as a specification. It constructs an **XrmValue** for the value string by calling **strlen**, which sets up the address and size. Then, it calls **XrmQPutResource**.

If the *database* argument is **NULL**, a new resource database is created and a pointer to the new resource database is returned in this argument. If the resource database is **NULL**, a new database is created.

¦ Copyright IBM Corp. 1985, 1991

2.4.285 XrmQuarkToString

char *XrmQuarkToString(*quark*)
XrmQuark *quark*;

quark Specifies the quark to be converted to a string.

XrmQuarkToString converts a quark to a string. The string, pointed to by the return value, must not be modified or freed.

#define XrmNameToString(*name***) XrmQuarkToString(***name***)**
#define XrmClassToString(*class***) XrmQuarkToString(***class***)**
#define XrmRepresentationToString(*type***) XrmQuarkToString**(*type*)

¦ Copyright IBM Corp. 1985, 1991

2.4.286 XrmStringToBindingQuarkList

XrmStringToBindingQuarkList(*string, bindings_return, quarks_return*)
char **string*;
XrmBindingList *bindings_return*;
XrmQuarkList *quarks_return*;

string Specifies the string to be converted.

bindings_return Returns the binding list.

quarks_return Returns the list of quarks.

XrmStringToBindingQuarkList converts a string with one or more components to a binding list and a quark list. Component names in the list are separated by a period (".") or an asterisk ("*") character. If the string does not start with a period or an asterisk, a period is assumed. For example, ***a.b*c** becomes:

quarks	a	b	c
bindings	loose	tight	loose

typedef enum {XrmBindTightly XrmBindLoosely} XrmBinding, *XrmBindingList;

A binding list, with **XrmBindingList** as *type*, indicates if components in name or class lists are bound tightly or loosely. If binding is loose, wildcarding indicates the component is specified.

XrmBindTightly indicates that a period (".") separates the components.

XrmBindLoosely indicates that an asterisk ("*") separates the components.

You must have sufficient space for the quarks list returned in *bindings_return* and *quarks_return* before calling **XrmStringToBindingQuarkList**.

¦ Copyright IBM Corp. 1985, 1991

2.4.287 XrmStringToQuark

XrmQuark XrmStringToQuark(*string*)
char **string*;

string Specifies the string to be converted into a quark.

XrmStringToQuark converts a string to a quark.

#define XrmStringToName(*string***) XrmStringToQuark(***string***)**
#define XrmStringToClass(*string***) XrmStringToClass(***string***)**
#define XrmStringToRepresentation(*string***) XrmStringToQuark**(*string*)

¦ Copyright IBM Corp. 1985, 1991

2.4.288 XrmStringToQuarkList

void XrmStringToQuarkList(*string, quarks_return*)
char **string*;
XrmQuarkList *quarks_return*;

string Specifies the string to be converted to a quark.

quarks_return Returns the list of quarks.

XrmStringToQuarkList converts a string with one or more components to a quark list. It converts the null-terminated *string*, which is generally a fully qualified name, to a list of quarks. The components of the string are separated by a period (".").

#define XrmStringToNameList(*string,name*)+
XrmStringToQuarkList((*string*),(*name*)
#define XrmStringToClassList(*string,class*)+
XrmStringToQuarkList((*string*),(*class*)

¦ Copyright IBM Corp. 1985, 1991

2.4.289 XrmUniqueQuark

XrmQuark XrmUniqueQuark()

XrmUniqueQuark allocates a new quark guaranteed not to represent any string.

¦ Copyright IBM Corp. 1985, 1991

2.4.290 XRotateBuffers

XRotateBuffers(*display, rotate*)
Display **display*;
int *rotate*;

display Specifies the connection to the X Server.

rotate Specifies how much to rotate the cut buffer.

XRotateBuffers rotates the cut buffers so that buffer zero becomes buffer *n*; buffer 1 becomes **n+1 mod 8**, and so on. This cut buffer numbering is global to the display. If any of the eight buffers has not been created, **XRotateBuffers** generates an error.

XRotateBuffers can generate the event errors **BadAtom**, **BadMatch**, and **BadWindow**.

¦ Copyright IBM Corp. 1985, 1991

2.4.291 XRotateWindowProperties

XRotateWindowProperties(*display, window, properties, num_prop, npositions*)
Display **display*;
Window *window*;
Atom *properties*[];
int *num_prop*;
int *npositions*;

display Specifies the connection to the X Server.

window Specifies the window ID.

properties Specifies the array of properties to be rotated.

num_prop Specifies the length of the properties array.

npositions Specifies the amount of the rotation.

XRotateWindowProperties rotates properties in the properties array. It rotates by *npositions* places around the virtual ring of property names (right for positive *npositions*, left for negative *npositions*).

If the property names in the properties array are viewed as being numbered starting with zero, there are *num_prop* property names in the list. The value associated with property name **I** becomes the value associated with property name **(I + npositions) mod N**, for **I** from zero to **N - 1**.

If *npositions mod N* is a non-zero, the X Server generates a **PropertyNotify** event for each property in the order listed in the array. See "Processing PropertyNotify Events" in topic 1.11.18.2.

If an atom occurs more than once in the list or a property name is undefined or does not exist, no properties are changed and an error is generated.

XRotateWindowProperties can generate the event errors **BadAtom**, **BadMatch**, and **BadWindow**.

¦ Copyright IBM Corp. 1985, 1991

2.4.292 XSaveContext

int XSaveContext(*display, window, context, data*)
Display **display*;
Window *window*;
XContext *context*;
caddr_t *data*;

display Specifies the connection to the X Server.

window Specifies the window with which the data is associated.

context Specifies the context type to which the data belongs.

data Specifies the data to be associated with the window and type.

XSaveContext saves a data value that corresponds to a window and context type. If an entry with the specified window and type already exists, it overrides that entry with the specified context. (If you know the entry already exists, use **XDeleteContext** first.)

If **XSaveContext** is successful, it returns a zero. Otherwise, it returns a non-zero.

XSaveContext can generate the error **XCNOMEM** (out of memory).

¦ Copyright IBM Corp. 1985, 1991

2.4.293 XSelectInput

XSelectInput(*display, window, event_mask*)
Display **display*;
Window *window*;
unsigned long *event_mask*;

display Specifies the connection to the X Server.

window Specifies the window ID.

event_mask Specifies the event mask. This mask is the bitwise inclusive OR of one or more of the valid event mask bits.

XSelectInput requests the X Server to report the events associated with the event masks in *event_mask*. Initially, X-Windows does not report these events. (See "Defining Event Masks" in topic 1.11.3.)

Events are reported relative to a window. If a window is not interested in an event, it usually propagates to the closest ancestor interested in the event, unless the *do_not_propagate* mask prohibits it.

A call to **XSelectInput** overrides any previous call to **XSelectInput** for the same window from the same client, but not for other clients. Different clients can select events on the same window with the following restrictions:

Multiple clients can select events on the same window because thei event masks are disjoint. After generating an event, the server reports the event to all interested clients.

Only one client at a time can select **CirculateRequest**, **ConfigureRequest**, or **MapRequest** events, which are associated with the event mask **SubstructureRedirectMask**.

Only one client at a time can select a **ResizeRequest** event, which is associated with the event mask **ResizeRedirectMask**.

Only one client at a time can select a **ButtonPress** event, which is associated with the event mask **ButtonPressMask**.

If a client passes both **ButtonPressMask** and **ButtonReleaseMask** for a specified window, a **ButtonPress** event in that window will automatically grab the mouse until all buttons are released and **ButtonRelease** events are sent to windows as described for **XGrabPointer**. This ensures that a window will see the **ButtonRelease** event corresponding to the **ButtonPress** event, even though the mouse may have exited the window in the meantime.

If a client passes **PointerMotionMask**, the X Server sends **MotionNotify** events independent of the state of the pointer buttons. If, instead, the client passes one or more of the event masks **Button1MotionMask**, **Button2MotionMask**, **Button3MotionMask**, **Button4MotionMask**, or **Button5MotionMask**, the X Server generates **MotionNotify** events only when one or more of the specified buttons is pressed. These masks are used to request **MotionNotify** events only when particular buttons are held down.

XSelectInput can generate the event errors **BadValue** and **BadWindow**.

¦ Copyright IBM Corp. 1985, 1991

2.4.294 XSendEvent

Status XSendEvent(*display, window, propagate, event_mask, event_send*)
Display **display*;
Window *window*;
Bool *propagate*;
unsigned long *event_mask*;
XEvent **event_send*;

display Specifies the connection to the X Server.

window Specifies the window ID of the window interested in the event and referred to as the destination window.

propagate Specifies a Boolean value.

event_mask Specifies the event mask. This mask is the bitwise inclusive OR of one or more of the valid event mask bits.

event_send Specifies a pointer to the event to be sent.

XSendEvent identifies the destination window, determines which clients should receive the specified events, and ignores any active grabs. This function requires an event mask. (See "Defining Event Masks" in topic 1.11.3.)

XSendEvent uses the *window* argument to identify the destination window. If this argument is:

PointerWindow, the destination window is the window that contains the pointer.

InputFocus, and the focus window contains the pointer, the destination window is the window that contains the pointer. If the focus window does not contain the pointer, the destination window is the focus window.

XSendEvent uses the *propagate* argument to determine which clients should receive the specified events. If this argument is:

False, the event is sent to every client selecting on destination any of the event types in the *event_mask* argument.

True, and no clients have selected on destination any of the event types in *event_mask*, the destination is replaced with the closest ancestor of destination for which some client has selected a type in *event_mask* and for which no intervening window has that type in its *do_not_propagate_mask*.

If no such window exists or if the window is an ancestor of the focus window and **InputFocus** was originally specified as the destination, the event is not sent to any clients.

Otherwise, the event is reported to every client selecting on the final destination any of the types specified in *event_mask*.

The events in the **XEvent** structure must be one of the core events or one of the events defined by a loaded extension, so that the X Server can swap byte contents correctly. Otherwise, the contents of the events are

¦ Copyright IBM Corp. 1985, 1991

unaltered and unchecked by the X Server, except to force *send_event* to **True** and to set the sequence number in the event correctly.

XSendEvent returns zero if the conversion-to-wire protocol failed. Otherwise, it returns a non-zero.

XSendEvent can generate the event errors **BadValue** and **BadWindow**.

¦ Copyright IBM Corp. 1985, 1991

2.4.295 XSetAccessControl

XSetAccessControl(*display, mode*)
Display **display*;
int *mode*;

display Specifies the connection to the X Server.

mode Specifies the mode (enable or disable).

XSetAccessControl enables or disables the use of the access control list at connection setups. If *mode* is:

EnableAccess, it enables host access control.

DisableAccess, it disables host access control.

The client and the X Server must reside on the same host or the client must have the required permission in the initial authorization at connection setup for **XSetAccessControl** to execute successfully. See "Controlling Host Access" in topic 1.10.6.

XSetAccessControl can generate the event errors **BadAccess** and **BadAlloc**.

¦ Copyright IBM Corp. 1985, 1991

2.4.296 XSetAfterFunction

int (*__XSetAfterFunction__(*display, proc*))()
Display **display*;
int (**proc*)();

display Specifies the connection to the X Server.

proc Specifies the function to be called after an **Xlib** function.

XSetAfterFunction sets the function to be called after another function that generates a protocol request is completed. It is called with a display pointer only.

¦ Copyright IBM Corp. 1985, 1991

2.4.297 XSetArcMode

XSetArcMode(*display*, *gc*, *arc_mode*)
Display **display*;
GC *gc*;
int *arc_mode*;

display Specifies the connection to the X Server.

gc Specifies the graphics context.

arc_mode Specifies the arc mode.

XSetArcMode sets the arc mode in the specified graphics context. If *arc_mode* is:

ArcChord, the arcs will be chord-filled.

ArcPieSlice, the arcs will be pie-sliced filled.

See "Manipulating Graphics Context or State" in topic 1.7.8.

XSetArcMode can generate the event errors **BadGC** and **BadValue**.

¦ Copyright IBM Corp. 1985, 1991

2.4.298 XSetBackground

XSetBackground(*display, gc, background*)
Display **display*;
GC *gc*;
unsigned long *background*;

display Specifies the connection to the X Server.

gc Specifies the graphics context.

background Specifies the background color to be set for the specified graphics context.

XSetBackground sets the background color in the specified graphics context. See "Creating Colormaps" in topic 1.7.1.

XSetBackground can generate the error **BadGC**.

¦ Copyright IBM Corp. 1985, 1991

2.4.299 XSetClassHint

XSetClassHint(*display, window, class_hints*)
Display **display*;
Window *window*;
XClassHint **class_hints*;

display Specifies the connection to the X Server.

window Specifies the window ID.

class_hints Specifies a pointer to a **XWMClassHint** structure to be used.

XSetClassHint sets the class hint for the specified window. See "Setting and Getting the Class of a Window" in topic 1.12.7.

XSetClassHint can generate the event errors **BadAlloc** and **BadWindow**.

¦ Copyright IBM Corp. 1985, 1991

2.4.300 XSetClipMask

XSetClipMask(*display, gc, pixmap*)
Display **display*;
GC *gc;*
Pixmap *pixmap;*

display Specifies the connection to the X Server.

gc Specifies the graphics context.

pixmap Specifies the pixmap.

XSetClipMask sets the *clip_mask* in the specified graphics context to the specified pixmap.

XSetClipMask can generate the event errors **BadGC**, **BadMatch**, and **BadValue**.

¦ Copyright IBM Corp. 1985, 1991

2.4.301 XSetClipOrigin

XSetClipOrigin(*display, gc, clip_x_origin, clip_y_origin*)
Display **display*;
GC *gc*;
int *clip_x_origin, clip_y_origin*;

display	Specifies the connection to the X Server.
gc	Specifies the graphics context.
clip_x_origin, clip_y_origin	Specifies the *x* and *y* coordinates of the clip origin.

XSetClipOrigin sets the clip origin in the specified graphics context.

XSetClipOrigin can generate the event error **BadGC**.

¦ Copyright IBM Corp. 1985, 1991

2.4.302 XSetClipRectangles

XSetClipRectangles(*display, gc, clip_x_origin, clip_y_origin, rectangles, n, c*
Display **display*;
GC *gc*;
int *clip_x_origin, clip_y_origin*;
XRectangle *rectangles*[];
int *n*;
int *ordering*;

display	Specifies the connection to the X Server.
gc	Specifies the graphics context.
clip_x_origin, clip_y_origin	Specifies the *x* and *y* coordinates of the clip origin.
rectangles	Specifies an array of rectangles for the graphics context.
n	Specifies the number of rectangles.
ordering	Specifies the ordering of the rectangles.

XSetClipRectangles changes the *clip_mask* in the specified graphics context to the specified list of *rectangles*.

The output is clipped to remain contained within the rectangles. The rectangle coordinates are interpreted relative to the origin of the clip. The rectangles should be non-intersecting or the graphics results will be undefined. The list of rectangles can be empty which disables output. (See "Drawing Points, Lines, Rectangles, and Arcs" in topic 1.8.3.)

The client can specify the ordering of the rectangles with the *ordering* argument as follows:

Unsorted indicates that the rectangles are in arbitrary order.

YSorted indicates that the rectangles are non-decreasing in their Y origin.

YXSorted, which constrains **YSorted**, indicates that all rectangles with an equal Y origin are non-decreasing in their X origin.

YXBanded, which constrains **YXSorted**, requires that for every possible Y scan line, all rectangles that include that scan line have an identical Y origins and Y extents.

XSetClipRectangles can generate the event errors **BadAlloc**, **BadGC**, **BadMatch**, and **BadValue**.

¦ Copyright IBM Corp. 1985, 1991

2.4.303 XSetCloseDownMode

XSetCloseDownMode(*display, close_mode*)
Display **display*;
int *close_mode*;

display Specifies the connection to the X Server.

close_mode Specifies the client closedown mode.

XSetCloseDownMode defines what will happen to the client's resources at connection close. Normally, a connection starts with the mode in **DestroyAll**.

The *close_mode* can also be set to **RetainPermanent** or **RetainTemporary**.

XSetCloseDownMode can generate the event error **BadValue**.

¦ Copyright IBM Corp. 1985, 1991

2.4.304 XSetCommand

XSetCommand(*display, window, argv, argc*)
Display **display*;
Window *window*;
char ***argv*;
int *argc*;

display Specifies the connection to the X Server.

window Specifies the window ID.

argv Specifies a pointer to the command and arguments used to start the application.

argc Specifies the number of arguments.

XSetCommand records the command and arguments used to invoke the application.

XSetCommand can generate event errors **BadWindow** and **BadAlloc**.

¦ Copyright IBM Corp. 1985, 1991

2.4.305 XSetDashes

XSetDashes(*display, gc, dash_offset, dash_list, n*)
Display **display*;
GC *gc;*
int *dash_offset*;
char *dash_list[];*
int *n*;

display	Specifies the connection to the X Server.
gc	Specifies the graphics context.
dash_offset	Specifies the phase of the pattern for the dashed line style for the specified graphics context.
dash_list	Specifies the dash list for the dashed line style for the specified graphics context.
n	Specifies the length of the dash list argument.

XSetDashes sets the *dash_offset* and *dash_list* for dashed line styles in the specified graphics context. Dashes cannot be empty. The initial and alternating elements of the *dash_list* are event dashes, and the others are the odd dashes. All of the elements must be non-zero.

Specifying an odd-length list is equivalent to specifying the same list concatenated with itself to produce an even-length list. See "Manipulating Graphics Context or State" in topic 1.7.8 for information on dashes, lines, and other graphics.

XSetDashes can generate the event errors **BadAlloc**, **BadGC**, and **BadValue**.

¦ Copyright IBM Corp. 1985, 1991

2.4.306 XSetErrorHandler

XSetErrorHandler(*handler*)
int (**handler*)(**Display***, **XErrorEvent***)

handler Specifies the program-supplied error handler.

XSetErrorHandler handles error events. It calls the program-supplied error handler whenever an **XError** event is received. This is not assumed to be a fatal condition. This error handler should not perform any operations (directly or indirectly) on the display.

¦ Copyright IBM Corp. 1985, 1991

2.4.307 XSetFillRule

XSetFillRule(*display, gc, fill_rule*)
Display **display*;
GC *gc;*
int *fill_rule;*

display Specifies the connection to the X Server.

gc Specifies the graphics context.

fill_rule Specifies the fill rule for the specified graphics context.

XSetFillRule sets the fill rule in the specified graphics context to **EvenOddRule** or **WindingRule**. See "Manipulating Graphics Context or State" in topic 1.7.8.

XSetFillRule can generate the event errors **BadGC** and **BadValue**.

¦ Copyright IBM Corp. 1985, 1991

2.4.308 XSetFillStyle

XSetFillStyle(*display, gc, fill_style*)
Display **display*;
GC *gc*;
int *fill_style*;

display Specifies the connection to the X Server.

gc Specifies the graphics context.

fill_style Specifies the fill style for the specified graphics context.

XSetFillStyle sets the fill style in the specified graphics context. The *fill_style* can be **FillSolid**, **FillTiled**, **FillStippled**, or **FillOpaqueStippled**. (See "Manipulating Graphics Context or State" in topic 1.7.8.)

XSetFillStyle can generate the event errors **BadGC** and **BadValue**.

¦ Copyright IBM Corp. 1985, 1991

2.4.309 XSetFont

XSetFont(*display*, *gc*, *font*)
Display **display*;
GC *gc*;
Font *font*;

display Specifies the connection to the X Server.

gc Specifies the graphics context.

font Specifies the font ID.

XSetFont sets the current font in the specified graphics context. See "Manipulating Fonts" in topic 1.9.

XSetFont can generate the event errors **BadAlloc**, **BadFont**, and **BadGC**.

¦ Copyright IBM Corp. 1985, 1991

2.4.310 XSetFontPath

XSetFontPath(*display, directories, ndirs*)
Display **display*;
char ***directories*;
int *ndirs*;

display Specifies the connection to the X Server.

directories Specifies the directory path to be used to look for a font. Setting the path to the empty list restores the default path defined for the X Server.

ndirs Specifies the number of directories in the path.

XSetFontPath defines the directory search path for looking for the font. There is a single search path per X Server, not per client. The interpretation of the directories is intended to specify directories to be searched in the order listed. Also, the contents of these strings are operating system-specific and are not intended to be used by client applications.

Usually, the X Server is free to cache font information internally rather than having to read fonts from files. The X Server is guaranteed to flush all cached information about fonts for which there currently are no explicit resource IDs allocated. An error from this request is system-specific.

XSetFontPath can generate the event error **BadValue**.

¦ Copyright IBM Corp. 1985, 1991

2.4.311 XSetForeground

XSetForeground(*display, gc, foreground*)
Display **display*;
GC *gc;*
unsigned long *foreground*;

display Specifies the connection to the X Server.

gc Specifies the graphics context.

foreground Specifies the foreground color for the specified graphics context.

XSetForeground sets the foreground color in the specified graphics context.

XSetForeground can generate the error **BadGC**.

¦ Copyright IBM Corp. 1985, 1991

2.4.312 XSetFunction

XSetFunction(*display, gc, function*)
Display **display*;
GC *gc;*
int *function*;

display Specifies the connection to the X Server.

gc Specifies the graphics context.

function Specifies the function for the specified graphics context.

XSetFunction sets a specified value in the specified graphics context.

XSetFunction can generate the event errors **BadGC** and **BadValue**.

¦ Copyright IBM Corp. 1985, 1991

2.4.313 XSetGraphicsExposures

XSetGraphicsExposures(*display, gc, graphics_exposures*)
Display **display*;
GC *gc*;
Boolean *graphics_exposures*;

display Specifies the connection to the X Server.

gc Specifies the graphics context.

graphics_exposures Specifies **GraphicsExpose** events to be reported when calling **XCopyArea** and **XCopyPlane** with this graphics context.

XSetGraphicsExposures sets the graphics-exposures flag in the specified graphics context. If *graphics_exposures* is:

True, **GraphicsExpose** events are reported.

False, **GraphicsExpose** events are not reported.

XSetGraphicsExposures can generate the event errors **BadGC** and **BadValue**.

¦ Copyright IBM Corp. 1985, 1991

2.4.314 XSetIconName

XSetIconName(*display, window, icon_name*)
Display **display*;
Window *window*;
char **icon_name*;

display Specifies the connection to the X Server.

window Specifies the window ID.

icon_name Specifies the icon name (a null-terminated string) of the window.

XSetIconName sets the name to be displayed in the icon window. This name is returned by **XGetIconName**.

XSetIconName can generate the event errors **BadAlloc** and **BadWindow**.

¦ Copyright IBM Corp. 1985, 1991

2.4.315 XSetIconSizes

XSetIconSizes(*display, window, size_list, count*)
Display **display*;
Window *window*;
XIconSize **size_list*;
int *count*;

display Specifies the connection to the X Server.

window Specifies the window ID.

size_list Specifies a pointer to the size list.

count Specifies the number of items in the size list.

XSetIconSizes sets the value of the icon size atom. It is used only by window managers to set the supported icon sizes. See "Setting and Getting Icon Sizing Hints" in topic 1.12.6.

XSetIconSizes can generate event errors **BadWindow** and **BadAlloc**.

¦ Copyright IBM Corp. 1985, 1991

2.4.316 XSetInputFocus

XSetInputFocus(*display, focus, revert_to, time*)
Display **display*;
Window *focus*;
int *revert_to*;
Time *time*;

display Specifies the connection to the X Server.

focus Specifies the window ID for the input focus.

revert_to Specifies the window ID for the window to which the the input focus reverts to if the specified window becomes unviewable.

time Specifies the time in a timestamp, which is expressed in milliseconds, or **CurrentTime**.

XSetInputFocus changes the input focus and the last-focus-change time. It has no effect if the specified time is earlier than the current last-focus-change time or is later than the current X Server time. Otherwise, the last-focus-change time is set to the specified time and the **CurrentTime** replaced by the current X Server time. The X Server generates **FocusIn** and **FocusOut** events.

Depending on the value assigned to the *focus* argument, **XSetInputFocus** executes as follows:

If **None** is specified, all keyboard events are discarded until a new focus window is set. In this case, the *revert_to* argument is ignored.

If a window ID is specified, it becomes the focus window for th keyboard. If a generated keyboard event would normally be reported to this window or one of its inferiors, the event is reported normally. Otherwise, the event is reported relative to the focus window.

If **PointerRoot** is specified, the focus window is dynamically taken to be the root window of whatever screen the pointer is on at each keyboard event. In this case, the *revert_to* argument is ignored.

If the specified focus window is not viewable at the time **XSetInputFocus** is called, an error is generated. If the focus window later becomes unviewable, the X Server evaluates the *revert_to* argument to determine the new focus window:

If **RevertToParent** is specified, the focus reverts to the parent or the closest viewable ancestor, and *revert_to* is set to **RevertToNone**.

If **RevertToPointerRoot** or **RevertToNone** is specified for *revert_to*, the focus reverts to that value. When the focus reverts, the X Server generates **FocusIn** and **FocusOut** events, but the last-focus-change time is not affected.

XSetInputFocus can generate the event errors **BadMatch**, **BadValue**, and **BadWindow**.

¦ Copyright IBM Corp. 1985, 1991

2.4.317 XSetIOErrorHandler

XSetIOErrorHandler(*handler*)
int (**handler*)(**Display** *);

handler Specifies the program-supplied error handler.

XSetIOErrorHandler sets the fatal IO error handler. The program-supplied error handler is called by **Xlib** if a system call error occurs, for example, if the connection to the server was lost. This is assumed to be a fatal condition. The routine should not return this condition. If the IO error handler does return, the client process will exit.

¦ Copyright IBM Corp. 1985, 1991

2.4.318 XSetLineAttributes

XSetLineAttributes(*display, gc, line_width, line_style, cap_style, join_style*)
Display **display*;
GC *gc*;
unsigned int *line_width*;
int *line_style*;
int *cap_style*;
int *join_style*;

display. Specifies the connection to the X Server.

gc Specifies the graphics context.

line_width Specifies the line width for the specified graphics context.

line_style Specifies the line style for the specified graphics context.

cap_style Specifies the line and cap style for the specified graphics context.

join_style Specifies the line-join style for the specified graphics context.

XSetLineAttributes sets the line drawing components in the specified graphics context. See "Manipulating Graphics Context or State" in topic 1.7.8.

The *line_style* variable can be **LineSolid**, **LineOnOffDash**, or **LineDoubleDash**.

The *cap_style* variable can be **CapNotLast**, **CapButt**, **CapRound**, or **CapProjecting**.

The *join_style* variable can be **JoinMiter**, **JoinRound**, or **JoinBevel**.

XSetLineAttributes can generate the event errors **BadGC** and **BadValue**.

¦ Copyright IBM Corp. 1985, 1991

2.4.319 XSetModifierMapping

int XSetModifierMapping(*display, modmap*)
Display **display*;
XModifierKeymap **modmap*;

display Specifies the connection to the X Server.

modmap Specifies a pointer to the **XModifierKeymap** structure.

XSetModifierMapping specifies the keycodes, if any, to be used as modifiers. If *modmap* is zero, no keycodes should be used. No two arguments can have the same non-zero keycode value.

The *modmap* member of the **XModifierKeymap** structure contains eight sets of *max_keypermod* keycodes, one for each modifier in the order **Shift, Lock, Control, Mod1, Mod2, Mod3, Mod4,** and **Mod5**. Only non-zero keycodes have meaning in each set, and non-zero keycodes are ignored. In addition, all non-zero keycodes must be in the range specified by *min_keycode* and *max_keycode* in the **Display** structure. No keycode may appear twice in the entire map. See "Manipulating Keyboard Encoding" in topic 1.10.5.

An X Server can impose restrictions on how modifiers can be changed. If restrictions are violated, the status reply is **MappingFailed** and none of the modifiers is changed.

If the new keycodes specified for a modifier differ from those currently defined and any current or new keys for that modifier are in the down state, the status reply is **MappingBusy**, and none of the modifiers is changed. **XSetModifierMapping** generates a **MappingNotify** event on a **MappingSuccess** status.

XSetModifierMapping can generate the event errors **BadAlloc** and **BadValue**.

¦ Copyright IBM Corp. 1985, 1991

2.4.320 XSetNormalHints

void XSetNormalHints(*display, window, hints*)
Display **display*;
Window *window*;
XSizeHints **hints*;

display Specifies the connection to the X Server.

window Specifies the window ID.

hints Specifies a pointer to the sizing hints for the window in its normal state.

XSetNormalHints sets the size hints structure for the specified window. Clients should use **XSetNormalHints** to inform the window manager of the size or position wanted for a window. A client's direct request to the X Server may be ignored by the window manager because window managers do not redirect configure requests, but they do pay attention to property changes.

An application must assign values to the appropriate elements in **XSizeHints** and set the *flags* field of this structure to indicate which information is present and where it came from. Otherwise, **XSetNormalHints** is meaningless. See "Setting and Getting Window Manager Sizing Hints" in topic 1.12.5.

XSetNormalHints can generate the event errors **BadAlloc** and **BadWindow**.

¦ Copyright IBM Corp. 1985, 1991

2.4.321 XSetPlaneMask

XSetPlaneMask(*display, gc, plane_mask*)
Display **display*;
GC *gc*;
unsigned long *plane_mask*;

display Specifies the connection to the X Server.

gc Specifies the graphics context.

plane_mask Specifies the plane mask.

XSetPlaneMask sets the plane mask in the specified graphics context. See "Manipulating Graphics Context or State" in topic 1.7.8.

XSetPlaneMask can generate the error **BadGC**.

¦ Copyright IBM Corp. 1985, 1991

2.4.322 XSetPointerMapping

int XSetPointerMapping(*display, map, nmap*)
Display **display*;
unsigned char *map[];*
int *nmap*;

display Specifies the connection to the X Server.

map Specifies the mapping list.

nmap Specifies the number of items in the mapping list.

XSetPointerMapping sets the mapping of the pointer. The X Server generates a **MappingNotify** event if the status is **MappingSuccess**.

If any of the buttons to be altered is currently in the down state, the status reply is **MappingBusy** and the mapping is not changed.

The elements of the list are indexed starting with 1. The length of the list indicates the number of physical buttons. The normal mapping for a pointer is the identify mapping:

```
map[i] = i
```

The index is a core button number and the element of the list defines the effective number. A zero element disables a button. Elements are not restricted in value by the number of physical buttons. However, no two elements can have the same non-zero value.

XSetPointerMapping can generate the event error **BadValue**.

¦ Copyright IBM Corp. 1985, 1991

2.4.323 XSetRegion

XSetRegion(*display, gc, region*)
Display **display*;
GC *gc;*
Region *region;*

display Specifies the connection to the X Server.

gc Specifies the graphics context.

region Specifies the region in which to set the specified graphics context.

XSetRegion sets the graphics contexts to the specified region. It sets the clip mask in the graphics contexts to the specified region. Once the clip mask has been set in the **GC**, the region can be destroyed. See "Manipulating Graphics Context or State" in topic 1.7.8.

¦ Copyright IBM Corp. 1985, 1991

2.4.324 XSetScreenSaver

XSetScreenSaver(*display, timeout, interval, prefer_blanking, allow_exposures*)
Display **display*;
int *timeout, interval*;
int *prefer_blanking*;
int *allow_exposures*;

display	Specifies the connection to the X Server.
timeout	Specifies the timeout, in seconds, until the screen saver turns on.
interval	Specifies the interval between screen saver invocations.
prefer_blanking	Specifies whether to enable screen blanking.
allow_exposures	Specifies the current screen saver control values.

XSetScreenSaver sets the screen saver. The *timeout* and *interval* are specified in seconds.

If *timeout* is non-zero, the screen saver is enabled. A *timeout* of zero disables the screen saver, while a *timeout* of -1 restores the default. Other negative values generate an event error.

An *interval* of zero disables the random pattern motion. If no input from devices (for example, keyboard, or mouse) is generated once the screen saver is enabled, for the specified number of timeout seconds, the screen saver is activated.

The variable *prefer_blanking* can be **DontPreferBlanking**, **PreferBlanking**, or **DefaultBlanking**.

For each screen, if blanking is preferred and the hardware supports video blanking, the screen will simply go blank. Otherwise, if exposures are allowed or the screen can be regenerated without sending exposure events to clients, the screen is tiled with the root window background tile. Otherwise, the state of the screens does not change, and the screen saver is not activated. The screen saver is deactivated and all screen states are restored at the next keyboard or pointer input or at the next call to **XForceScreenSaver** with mode **ScreenSaverReset**.

The variable *allow_exposure* can be **DontAllowExposures**, **AllowExposures**, or **DefaultExposures**.

XSetScreenSaver can generate the event error **BadValue**.

¦ Copyright IBM Corp. 1985, 1991

2.4.325 XSetSelectionOwner

XSetSelectionOwner(*display*, *selection*, *owner*, *time*)
Display **display*;
Atom *selection*;
Window *owner*;
Time *time*;

display Specifies the connection to the X Server.

selection Specifies the selection atom.

owner Specifies the owner of the specified selection atom with a window ID or **None**.

time Specifies the time in a timestamp, which is expressed in milliseconds, or **CurrentTime**.

XSetSelectionOwner sets the selection owner. It changes the owner and last-change time for the specified selection. It has no effect if *time* is earlier than the current last-change time of the specified selection or is later than the current X Server time. Otherwise, the last-change time is set to the current server time.

If *owner* is **None**, the selection will have no owner. Otherwise, the owner of the selection becomes the client executing the request.

If the new owner is not the same as the current owner of the selection and the current owner is not **None**, the current owner is sent a **SelectionClear** event.

If the client is the owner of a selection and is terminated, or if the owner window specified in the request is destroyed, the owner of the selection automatically reverts to **None**, but the last-change time is not affected. The selection atom is uninterpreted by the X Server. The owner window is returned by **XGetSelectionOwner** and is reported in **SelectionRequest** and **SelectionClear** events. Selections are global to the X Server.

XSetSelectionOwner can generate the event errors **BadAtom** and **BadWindow**.

¦ Copyright IBM Corp. 1985, 1991

2.4.326 XSetSizeHints

XSetSizeHints(*display, window, hints, property*)
Display **display*;
Window *window*;
XSizeHints **hints*;
Atom *property*;

display Specifies the connection to the X Server.

window Specifies the window ID.

hints Specifies a pointer to the size hints.

property Specifies the property atom.

XSetSizeHints sets the value of any property of type WM_SIZE_HINTS. It sets the **XSizeHints** structure for the property and the window. See "Setting and Getting Window Manager Sizing Hints" in topic 1.12.5.

XSetSizeHints can generate the event errors **BadAlloc**, **BadAtom**, and **BadWindow**.

¦ Copyright IBM Corp. 1985, 1991

2.4.327 XSetStandardColormap

void XSetStandardColormap(*display, window, cmap, property*)
Display **display*;
Window *window*;
XStandardColormap **cmap*;
Atom *property*;

display Specifies the connection to the X Server.

window Specifies the window ID.

cmap Specifies the colormap ID.

property Specifies the property atom.

XSetStandardColormap creates or changes a standard colormap. For more information on standard colormaps, see "Manipulating Standard Colormaps" in topic 1.7.3.

This function is used by window managers usually to create a standard colormap with the following procedure:

1. Grab the server. See **XGrabServer** in this chapter.
2. See if the property is on the property list of the root window for the display. If the desired property is not present, do the following:
   Create a colormap (not required for **RGB_DEFAULT_MAP**)
   Determine the color capabilities of the display
   Call **XAllocColorPlanes** or **XAllocColorCells** to allocate cells in the colormap
   Call **XStoreColors** to store appropriate color values in the colormap
   Fill in the descriptive fields in the property
   Attach the property to the root window.
3. Ungrab the server.

XSetStandardColormap can generate the event errors **BadAlloc**, **BadAtom**, and **BadWindow**.

¦ Copyright IBM Corp. 1985, 1991

2.4.328 XSetStandardProperties

XSetStandardProperties(*display, window, window_name, icon_name, icon_pixmap, argv, argc, hints*)

```
Display *display;
Window window;
char *window_name;
char *icon_name;
Pixmap icon_pixmap;
char **argv;
int argc;
XSizeHints *hints;
```

display	Specifies the connection to the X Server.
window	Specifies the window ID.
window_name	Specifies the window name.
icon_name	Specifies the name to be displayed in the icon window.
icon_pixmap	Specifies the pixmap or **None** for the icon window.
argv	Specifies a pointer to the command and arguments used to start the application.
argc	Specifies the number of arguments.
hints	Specifies a pointer to the sizing hints for the window in its normal state.

XSetStandardProperties specifies a minimum set of properties with a single call. It provides information to the window manager about your program. This function sets all or portions of the WM_NAME, WM_ICON_NAME, WM_HINTS, WM_COMMAND, and WM_NORMAL_HINTS properties. (See "Setting and Getting Window Manager Sizing Hints" in topic 1.12.5.)

XSetStandardProperties can generate the event errors **BadAlloc** and **BadWindow**.

¦ Copyright IBM Corp. 1985, 1991

2.4.329 XSetState

XSetState(*display, gc, foreground, background, function, plane_mask*)
Display **display*;
GC *gc*;
unsigned long *foreground*, *background*;
int *function*;
unsigned long *plane_mask*;

display Specifies the connection to the X Server.

gc Specifies the graphics context.

foreground Specifies the foreground for the specified graphics context.

background Specifies the background for the specified graphics context.

function Specifies the function for the specified graphics context.

plane_mask Specifies the plane mask.

XSetState sets the foreground, background, plane mask, and function components for the specified graphics context. See "Manipulating Graphics Context or State" in topic 1.7.8.

XSetState can generate the event errors **BadGC** and **BadValue**.

¦ Copyright IBM Corp. 1985, 1991

2.4.330 XSetStipple

XSetStipple(*display, gc, stipple*)
Display **display*;
GC *gc*;
Pixmap *stipple*;

display Specifies the connection to the X Server.

gc Specifies the graphics context.

stipple Specifies the stipple for the specified graphics context.

XSetStipple sets the stipple in the specified graphics context. The depth must be 1. Some displays have the hardware support for stippling.

XSetStipple can generate the event errors **BadAlloc**, **BadGC**, **BadPixmap** and **BadMatch**.

¦ Copyright IBM Corp. 1985, 1991

2.4.331 XSetSubwindowMode

XSetSubwindowMode(*display, gc, subwindow_mode*)
Display **display*;
GC *gc;*
int *subwindow_mode*;

display	Specifies the connection to the X Server.
gc	Specifies the graphics context.
subwindow_mode	Specifies the subwindow mode.

XSetSubwindowMode sets the subwindow mode in the specified graphics context. The *subwindow_mode* can be one of the following:

ClipByChildren, which clips source and destination by all viewable children

IncludeInferiors, which draws through all subwindows.

See also "Manipulating Graphics Context or State" in topic 1.7.8.

XSetSubwindowMode can generate the event errors **BadGC** and **BadValue**.

¦ Copyright IBM Corp. 1985, 1991

2.4.332 XSetTile

XSetTile(*display*, *gc*, *tile*)
Display **display*;
GC *gc*;
Pixmap *tile*;

display Specifies the connection to the X Server.

gc Specifies the graphics context.

tile Specifies the fill tile to be set for the specified graphics context.

XSetTile sets the fill tile in the specified graphics context. The depth of the tile must be the same as the depth of the screen. See "Manipulating Graphics Context or State" in topic 1.7.8.

XSetTile can generate the event errors **BadAlloc**, **BadGC**, **BadMatch**, and **BadPixmap**.

¦ Copyright IBM Corp. 1985, 1991

2.4.333 XSetTransientForHint

XSetTransientForHint(*display, window, prop_window*)
Display **display*;
Window *window*;
Window *prop_window*;

display Specifies the connection to the X Server.

window Specifies the window ID.

prop_window Specifies the window ID for the WM_TRANSIENT_FOR property.

XSetTransientForHint sets the WM_TRANSIENT_FOR atom of the specified window to the specified *prop_window*. It indicates that a transient top-level window is not really a full-fledged window. Instead, the transient top-level window is operating on behalf of another window. **XSetTransientForHint** sets the property to be the transient window of the main window. See "Setting and Getting the Transient Property" in topic 1.12.8.

XSetTransientForHint can generate the event errors **BadAlloc** and **BadWindow**.

¦ Copyright IBM Corp. 1985, 1991

2.4.334 XSetTSOrigin

XSetTSOrigin(*display, gc, ts_x_origin, ts_y_origin*)
Display **display*;
GC *gc*;
int *ts_x_origin, ts_y_origin*;

display	Specifies the connection to the X Server.
gc	Specifies the graphics context.
ts_x_origin, ts_y_origin	Specifies the *x* and *y* coordinates of the tile or stipple origin.

XSetTSOrigin sets the tile or stipple origin in the specified graphics context.

XSetTSOrigin can generate the event error **BadGC**.

¦ Copyright IBM Corp. 1985, 1991

2.4.335 XSetWindowBackground

XSetWindowBackground(*display, window, background_pixel*)
Display **display*;
Window *window*;
unsigned long *background_pixel*;

display	Specifies the connection to the X Server.
window	Specifies the window ID.
background_pixel	Specifies the pixel of the background that determines what entry in the colormap to use.

XSetWindowBackground sets the background pixel of the window to the pixel value specified. It uses a pixmap of undefined size filled with the color associated with the pixel value in the *background_pixel* argument. **XSetWindowBackground** cannot be used on an **InputOnly** window.

XSetWindowBackground can generate the errors **BadMatch** and **BadWindow**.

Note: **XSetWindowBackground** does not change the current contents of the window. Clear and repaint the screen after this function completes.

¦ Copyright IBM Corp. 1985, 1991

2.4.336 XSetWindowBackgroundPixmap

XSetWindowBackgroundPixmap(*display, window, background_pixmap*)
Display **display*;
Window *window*;
Pixmap *background_pixmap*;

display Specifies the connection to the X Server.

window Specifies the window ID.

background_pixmap Specifies the background pixmap.

XSetWindowBackgroundPixmap sets the background pixmap of the window to the pixmap specified.

If *background_pixmap* is:

A pixmap ID, the background is painted with this pixmap
None, no background is painted.
ParentRelative, the pixmap of the parent is used.

If no background pixmap is specified, the background pixmap of the parent window is used. On the root window, the default background will be restored. The background pixmap can be freed if no further explicit references to it are made.

XSetWindowBackgroundPixmap cannot be used on an **InputOnly** window.

XSetWindowBackgroundPixmap can generate the event errors **BadColor**, **BadMatch**, **BadPixmap**, and **BadWindow**.

Note: **XSetWindowBackgroundPixmap** does not change the current contents of the window. Clear and repaint the screen after this function is completes.

¦ Copyright IBM Corp. 1985, 1991

2.4.337 XSetWindowBorder

XSetWindowBorder(*display, window, borderpixel*)
Display **display*;
Window *window*;
unsigned long *borderpixel*;

display Specifies the connection to the X Server.

window Specifies the window ID.

borderpixel Specifies the entry in the colormap.

XSetWindowBorder sets the border pixel of the window to the pixel value specified. This value is used to determine what color to paint the border.

XSetWindowBorder cannot be used on an **InputOnly** window.

XSetWindowBorder can generate the event errors **BadMatch**, **BadPixmap**, **BadValue**, and **BadWindow**.

¦ Copyright IBM Corp. 1985, 1991

2.4.338 XSetWindowBorderPixmap

XSetWindowBorderPixmap(*display, window, borderpixmap*)
Display **display*;
Window *window*;
Pixmap *borderpixmap*;

display Specifies the connection to the X Server.

window Specifies the window ID.

borderpixmap Specifies the border pixmap.

XSetWindowBorderPixmap sets the border pixmap of the window to the pixmap specified. The border pixmap can be freed immediately if no further explicit references to it are to be made.

If *borderpixmap* is:

A pixmap ID, the associated pixmap is used for the border

CopyFromParent, a copy of the border pixmap of the parent window is used.

XSetWindowBorderPixmap cannot be used on an **InputOnly** window.

XSetWindowBorderPixmap can generate the errors **BadMatch**, **BadPixmap**, **BadValue**, and **BadWindow**.

¦ Copyright IBM Corp. 1985, 1991

2.4.339 XSetWindowBorderWidth

XSetWindowBorderWidth(*display, window, width*)
Display * *display*;
Window *window*;
unsigned int *width*;

display Specifies the connection to the X Server.

window Specifies the window ID.

width Specifies the width for the window border.

XSetWindowBorderWidth sets the width of the specified window border.

XSetWindowBorderWidth can generate the errors **BadValue** and **BadWindow**.

¦ Copyright IBM Corp. 1985, 1991

2.4.340 XSetWindowColormap

XSetWindowColormap(*display, window, cmap*)
Display **display*;
Window *window*;
Colormap *cmap*;

display Specifies the connection to the X Server.

window Specifies the window ID for the window where the colormap is to be set.

cmap Specifies the colormap ID.

XSetWindowColormap sets the colormap for a specified window. See "Creating Colormaps" in topic 1.7.1.

XSetWindowColormap can generate the event errors **BadColor**, **BadMatch**, and **BadWindow**.

¦ Copyright IBM Corp. 1985, 1991

2.4.341 XSetWMHints

XSetWMHints(*display, window, wmhints*)
Display **display*;
Window *window*;
XWMHints **wmhints*;

display Specifies the connection to the X Server.

window Specifies the window ID.

wmhints Specifies a pointer to the window manager hints.

XSetWMHints sets the window manager hints that include icon information and location, the initial state of the window, and if the application relies on the window manager to get keyboard input. See "Setting and Getting Window Manager Hints" in topic 1.12.4.

XSetWMHints can generate the event errors **BadAlloc** and **BadWindow**.

¦ Copyright IBM Corp. 1985, 1991

2.4.342 XSetZoomHints

XSetZoomHints(*display, window, zhints*)
Display **display*;
Window *window*;
XSizeHints **zhints*;

display Specifies the connection to the X Server.

window Specifies the window ID.

zhints Specifies a pointer to the zoom hints.

XSetZoomHints provides the window manager with information for the window in the zoomed state. It sets and reads the WM_ZOOM_HINTS property. See "Setting and Getting Window Manager Sizing Hints" in topic 1.12.5.

XSetZoomHints can generate event errors **BadAlloc** and **BadWindow**.

¦ Copyright IBM Corp. 1985, 1991

2.4.343 XShrinkRegion

XShrinkRegion(*region, dx, dy*)
Region *region*;
int *dx, dy*;

region Specifies the region.

dx, dy Specifies the *x* and *y* coordinates for the amount by which to reduce the specified region.

XShrinkRegion reduces the specified region by a specified amount. Positive values reduce the size of the region, while negative values increase the region.

¦ Copyright IBM Corp. 1985, 1991

2.4.344 XStoreBuffer

XStoreBuffer(*display, bytes, nbytes, buffer*)
Display **display*;
char *bytes*[];
int *nbytes*;
int *buffer*;

display Specifies the connection to the X Server.

bytes Specifies the string of bytes to be stored which is not necessarily an ASCII string or a null-terminated string.

nbytes Specifies the number of bytes of the *bytes* argument to be stored.

buffer Specifies the buffer in which to store the byte string.

XStoreBuffer stores data in a specified cut buffer.

This routine can generate the event errors **BadAlloc**, **BadAtom**, and **BadWindow**.

¦ Copyright IBM Corp. 1985, 1991

2.4.345 XStoreBytes

XStoreBytes(*display, bytes, nbytes*)
Display **display*;
char *bytes*[];
int *nbytes*;

display	Specifies the connection to the X Server.
bytes	Specifies the string of bytes to be stored which is not necessarily an ASCII string or a null-terminated string.
nbytes	Specifies the number of bytes to be stored.

XStoreBytes stores data in cut buffer zero. It returns the number of bytes to be stored in the *nbytes* argument. The contents of the cut buffer do not need to be text. The contents of the buffer may be retrieved later with **XFetchBytes**.

XStoreBytes can generate the event errors **BadAlloc** and **BadWindow**.

¦ Copyright IBM Corp. 1985, 1991

2.4.346 XStoreColor

XStoreColor(*display, cmap, defs*)
Display **display*;
Colormap *cmap*;
XColor **defs*;

display Specifies the connection to the X Server.

cmap Specifies the colormap ID.

defs Specifies a color definition structure.

XStoreColor changes the colormap entry of the pixel value specified in the pixel member of the **XColor** structure. (See "Creating Colormaps" in topic 1.7.1.) **XStoreColor** changes:

The colormap entry of the pixel value specified in the pixel member o the **XColor** structure. This pixel value must be a read/write cell and a valid index in the colormap.

The red, green, and/or blue color components specified by indicatin **DoRed**, **DoGreen**, and/or **DoBlue** to the *flags* member of the **XColor** structure. If the colormap is an installed colormap for the screen, the changes are visible immediately.

The specified pixel if it is allocated writable in *cmap* by any client, even if the pixel generates an error.

XStoreColor can generate the event errors **BadColor** and **BadValue**

¦ Copyright IBM Corp. 1985, 1991

2.4.347 XStoreColors

XStoreColors(*display, cmap, defs, ncolors*)
Display **display*;
Colormap *cmap*;
XColor *defs*[];
int *ncolors*;

display Specifies the connection to the X Server.

cmap Specifies the colormap ID.

defs Specifies a color definition structure.

ncolors Specifies the number of **XColor** structures in the color definition array.

XStoreColors changes the colormap entries of the pixel values specified in the pixel members of the **XColor** structures. (See "Creating Colormaps" in topic 1.7.1.)

Specify the color components by indicating **DoRed**, **DoGreen**, and/or **DoBlue** to the *flags* member of the **XColor** structures. If the colormap is an installed map for its screen, the changes are visible immediately.

XStoreColors changes the specified pixels if they are allocated writable in *cmap* by any client, even if one or more pixels generates an error.

XStoreColors can generate the event errors **BadAccess**, **BadColor**, and **BadValue**.

¦ Copyright IBM Corp. 1985, 1991

2.4.348 XStoreName

XStoreName(*display, window, window_name*)
Display **display*;
Window *window*;
char **window_name*;

display Specifies the connection to the X Server.

window Specifies the window ID

window_name Specifies the window name as a null-terminated string.

XStoreName assigns the name passed to *window_name* to the specified window. This name is returned with any subsequent call to **XFetchName**.

A window manager may display the window name in a prominent place, such as the titlebar, so that users can identify the windows. Other window managers display the icon window name as the window name. Displaying the icon window name is not encouraged.

XStoreName can generate the event errors **BadAlloc** and **BadWindow**.

¦ Copyright IBM Corp. 1985, 1991

2.4.349 XStoreNamedColor

XStoreNamedColor(*display, cmap, color, pixel, flags*)
Display **display*;
Colormap *cmap*;
char **color*;
unsigned long *pixel*;
int *flags*;

display	Specifies the connection to the X Server.
cmap	Specifies the colormap ID.
color	Specifies the color name string, which is case-sensitive, for the color cell.
pixel	Specifies the entry in the colormap.
flags	Specifies which red, green, and blue indexes are set.

XStoreNamedColor allocates a color cell by name. It searches for the color with respect to the screen associated with *cmap*. It stores the result in *cmap*.

The *flags* argument can be bitwise inclusive OR of the bits from the constant set **DoRed**, **DoGreen**, and **DoBlue**.

XStoreNamedColor can generate the event errors **BadAccess**, **BadColor**, **BadName**, and **BadValue**.

¦ Copyright IBM Corp. 1985, 1991

2.4.350 XStringToKeysym

KeySym XStringToKeysym(*string*)
char **string*;

string Specifies the name of the keysym to be converted.

XStringToKeysym converts the keysym name to the keysym code. Valid **keysym** names are listed in **<X11/keysymdef.h>**. If the specified string does not match a valid keysym, **XStringToKeysym** returns **NoSymbol**.

¦ Copyright IBM Corp. 1985, 1991

2.4.351 XSubImage

XImage *XSubImage(*ximage, x, y, subimage_width, subimage_height*)
XImage **ximage*;
int *x*;
int *y*;
int *subimage_width*;
int *subimage_height*;

ximage	Specifies a pointer to the image.
x, y	Specifies the *x* and *y* coordinates.
subimage_width	Specifies the width of the new subimage in pixels.
subimage_height	Specifies the height of the new subimage in pixels.

XSubImage creates a new image that is a subsection of an existing image. It allocates the memory necessary for the new **XImage** structure. It returns a pointer to the new image. See "Transferring Images Between Client and Server" in topic 1.9.3.

XSubImage uses repetitive calls to **XGetPixel** and **XPutPixel**.

¦ Copyright IBM Corp. 1985, 1991

2.4.352 XSubtractRegion

XSubtractRegion(*sra, srb, dr*)
Region *sra, srb, dr*;

sra, srb Specifies the two regions for the computation.

dr Stores the result of the computation.

XSubtractRegion subtracts the region in *srb* from the region in *sra*, and then stores the result in *dr*.

¦ Copyright IBM Corp. 1985, 1991

2.4.353 XSync

XSync(*display, discard*)
Display **display*;
int *discard*;

display Specifies the connection to the X Server.

discard Specifies whether or not to discard all events on the event queue.

XSync flushes the output buffer. Then, it waits until all events, including error events, have been received and processed by the X Server.

Errors generated must be handled by the error handler. For each error event received and processed by the X Server, **XSync** calls **XError**.

If *discard* is:

Zero, **XSync** does not discard the events on the queue.
One, **XSync** discards all events on the queue, including those events that were on the queue before it was called.

Client applications seldom need to call **XSync**.

¦ Copyright IBM Corp. 1985, 1991

2.4.354 XSynchronize

int (*__XSynchronize__(*display, onoff*))()
Display **display*;
int *onoff*;

display Specifies the connection to the X Server.

onoff Specifies whether to enable or disable synchronization.

XSynchronize returns the previous after function. If *onoff* is:

Zero, **XSynchronize** disables synchronization or sets synchronization to off.
Non-zero, **XSynchronize** sets synchronization to on.

¦ Copyright IBM Corp. 1985, 1991

2.4.355 XTextExtents

XTextExtents(*font_struct, string, nchars, direction_return, ascent_return, descent_return, overall_return*)
XFontStruct **font_struct*;
char **string*;
int *nchars*;
int **direction_return*;
int **ascent_return, *descent_return*;
XCharStruct **overall_return*;

font_struct Specifies a pointer to the **XFontStruct** structure.

string Specifies the character string.

nchars Specifies the number of characters in the character string.

direction_return Returns the value of the direction (**FontLeftToRight** or **FontRightToLeft**) hint member.

ascent_return Returns the font ascent member which is the maximum of the ascent metrics of all characters in the string.

descent_return Returns the font descent member which is the maximum of the descent metrics.

overall_return Returns the overall size in the specified **XCharStruct** structure.

XTextExtents determines the logical extents of the specified 8-bit character string. The logical extents of a string are the width and height of the bounding box occupied by the string in the specified font. **XTextExtents** performs the size computation locally.

XTextExtents returns an **XCharStruct** structure with *width* set to the sum of the character-width metrics of all characters in the string. For each character in the string:

W should be the sum of the character-width metrics of all characters preceding it in the string.
R should be the right-side-bearing metric of the character plus *W*.
The *lbearing* member is set to the the minimum *L* of all characters in the string.
The *rbearing* member is set to the maximum *R*.

See "Manipulating Fonts" in topic 1.9.

Use **XQueryTextExtents** to query the server for the sizes of an 8-bit character string.

¦ Copyright IBM Corp. 1985, 1991

2.4.356 XTextExtents16

XTextExtents16(*font_struct, string, nchars, direction_return, ascent_return, descent_return, overall_return*)
XFontStruct **font_struct*;
XChar2b **string*;
int *nchars*;
int **direction_return*;
int **ascent_return, *descent_return*;
XCharStruct **overall_return*;

font_struct Specifies a pointer to the **XFontStruct** structure.

string Specifies the character string.

nchars Specifies the number of characters in the character string.

direction_return Returns the value of the direction (**FontLeftToRight** or **FontRightToLeft**) hint member.

ascent_return Returns the font ascent member which is the maximum of the ascent metrics of all characters in the string.

descent_return Returns the font descent member which is the maximum of the descent metrics.

overall_return Returns the overall size in the specified **XCharStruct** structure.

XTextExtents16 returns the logical extents of the specified 2-byte character string. It performs the size computation locally.

XTextExtents16 returns an **XCharStruct** structure with *width* set to the sum of the character-width metrics of all characters in the string. For each character in the string:

W should be the sum of the character-width metrics of all characters preceding it in the string.
R should be the right-side-bearing metric of the character plus *W*.
The *lbearing* member is set to the the minimum *L* of all characters in the string.
The *rbearing* member is set to the maximum *R*.

See "Manipulating Fonts" in topic 1.9.

If the font has no defined default character, undefined characters in the string are zero.

Use **XQueryTextExtents16** to query the server for the sizes of a 16-bit character string.

¦ Copyright IBM Corp. 1985, 1991

2.4.357 XTextWidth

int XTextWidth(*font_struct, string, count*)
XFontStruct **font_struct*;
char **string*;
int *count*;

font_struct Specifies the font used for the width computation.

string Specifies the character string.

count Specifies the character count in the named string.

XTextWidth determines the width of an 8-bit character string. Width is computed by adding the character widths of all of the characters. **XTextWidth** returns the sum of the character metrics in pixels. See "Manipulating Fonts" in topic 1.9.

¦ Copyright IBM Corp. 1985, 1991

2.4.358 XTextWidth16

int XTextWidth16(*font_struct, string, count*)
XFontStruct **font_struct*;
XChar2b **string*;
int *count*;

font_struct Specifies the font used for the width computation.

string Specifies the character string.

count Specifies the character count in the string.

XTextWidth16 determines the width of a 2-byte character string. Width is computed by adding the character widths of all of the characters. **XTextWidth16** returns the sum of the character metrics in pixels. See "Manipulating Fonts" in topic 1.9.

¦ Copyright IBM Corp. 1985, 1991

2.4.359 XTranslateCoordinates

int XTranslateCoordinates(*display, src_window, dest_window, src_x, src_y, dest*
dest_y_return, child_return)
Display **display*;
Window *src_window, dest_window*;
int *src_x, src_y*;
int **dest_x_return, *dest_y_return*;
Window **child_return*;

display	Specifies the connection to the X Server.
src_window	Specifies the window ID of the source window.
dest_window	Specifies the window ID of the destination window.
src_x, src_y	Specifies the *x* and *y* coordinates within the source window.
dest_x_return, dest_y_return	Returns the *x* and *y* coordinates within the destination window.
child_return	Returns the child window if the coordinates are contained in a mapped child of the destination window.

XTranslateCoordinates performs a coordinate transformation from the coordinate space of one window to another window, or it determines which subwindow contains a coordinate.

XTranslateCoordinates takes the *src_x* and *src_y* coordinates (relative to the origin of the source window) within the source window. It returns these coordinates (relative to the origin of the destination window) to *dest_x_return* and *dest_y_return*.

If **XTranslateCoordinates** returns zero, it indicates that *src_w* and *dest_w* are on different screens and *dest_x_return* and *dest_y_return* are zero.

If the coordinates are contained in a mapped child of *dest_w*, that child is returned to the *child_return* argument.

XTranslateCoordinates can generate the event error **BadWindow**.

¦ Copyright IBM Corp. 1985, 1991

2.4.360 XUndefineCursor

XUndefineCursor(*display, window*)
Display **display*;
Window *window*;

display Specifies the connection to the X Server.

window Specifies the window ID.

XUndefineCursor undefines the cursor in the window. When the mouse is in the window, the cursor of the parent window is used. The default cursor is restored on the root window with no cursor specified.

This routine can generate the event error **BadWindow**.

¦ Copyright IBM Corp. 1985, 1991

2.4.361 XUngrabButton

XUngrabButton(*display, button_ungrab, modifiers, ungrab_window*)
Display **display*;
unsigned int *button_ungrab*;
unsigned int *modifiers*;
Window *ungrab_window*;

display	Specifies the connection to the X Server.
button_ungrab	Specifies the pointer button that is to be released in combination with the modifier keys.
modifiers	Specifies the set of keymasks. This mask is the bitwise inclusive OR of valid keymask bits.
ungrab_window	Specifies the window ID of the window to be ungrabbed.

XUngrabButton ungrabs a mouse button. It releases the button/key combination on the specified window if it was grabbed by this client. This request fails if another client has already issued a **XGrabButton** with the same button/key combination on the same window.

The variable *button_ungrab* can be set to **AnyButton**, which is equivalent to issuing the ungrab request for all possible buttons. This request has no effect on an active grab.

The *modifiers* can be one of the following valid keymask bits:

ShiftMask	**Mod1Mask**	**Mod4Mask**
LockMask	**Mod2Mask**	**Mod5Mask**
ControlMask	**Mod3Mask**	

Or, it can be set to **AnyModifier**, which is equivalent to issuing the ungrab request for all possible modifier combinations, including the combination of no modifiers. This request has no effect on an active grab. See "Grabbing the Pointer" in topic 1.10.2.

XUngrabButton can generate the event error **BadWindow**.

¦ Copyright IBM Corp. 1985, 1991

2.4.362 XUngrabKey

XUngrabKey(*display, keycode, modifiers, ungrab_window*)
Display **display*;
int *keycode*;
unsigned int *modifiers*;
Window *ungrab_window*;

display	Specifies the connection to the X Server.
keycode	Specifies the keycode or **AnyKey** which maps to the specific key to be ungrabbed.
modifiers	Specifies the set of keymasks. This mask is the bitwise inclusive OR of valid keymask bits.
ungrab_window	Specifies the window ID of the window associated with the keys to be ungrabbed.

XUngrabKey ungrabs a key. It releases the key combination on the specified window if it was grabbed by this client. It has no effect on an active grab.

The variable *modifiers* can be set to the following valid keymask bits:

ShiftMask	**Mod1Mask**	**Mod4Mask**
LockMask	**Mod2Mask**	**Mod5Mask**
ControlMask	**Mod3Mask**	

Or, it can be set to **AnyModifier**, which is equivalent to issuing the ungrab key request for all possible modifier combinations. See "Grabbing the Pointer" in topic 1.10.2.

XUngrabkey can generate the event error **BadWindow**.

¦ Copyright IBM Corp. 1985, 1991

2.4.363 XUngrabKeyboard

XUngrabKeyboard(*display, time*)
Display **display*;
Time *time;*

display Specifies the connection to the X Server.

time Specifies the time in a timestamp, which is expressed in milliseconds, or **CurrentTime**.

XUngrabKeyboard releases the keyboard and any queued events if the client has actively grabbed it with **XGrabKeyboard** or **XGrabKey**. If the specified time is earlier than the last-keyboard-grab time or is later than the current X Server time, **XUngrabKeyboard** does not release the keyboard and any queued events.

XUngrabKeyboard generates **FocusIn** and **FocusOut** events. The X Server automatically performs an **XUngrabKeyboard** if the event window for an active keyboard grab becomes unviewable. See "Grabbing the Pointer" in topic 1.10.2.

¦ Copyright IBM Corp. 1985, 1991

2.4.364 XUngrabPointer

XUngrabPointer(*display, time*)
Display **display*;
Time *time;*

display Specifies the connection to the X Server.

time Specifies the time in a timestamp, which is expressed in milliseconds, or **CurrentTime**.

XUngrabPointer releases the pointer and any queued events, if this client has actively grabbed the pointer with **XGrabPointer**, **XGrabButton**, or from a normal button press. If the specified time is earlier than the last-pointer-grab time or is later than the current X Server time, this function does not release the pointer.

XUngrabPointer also generates **EnterNotify** and **LeaveNotify** events. If the event window or confine-to window for an active pointer grab becomes unviewable, the X Server performs an **XUngrabPointer** automatically. See "Grabbing the Pointer" in topic 1.10.2.

¦ Copyright IBM Corp. 1985, 1991

2.4.365 XUngrabServer

XUngrabServer(*display*)
Display **display*;

display Specifies the connection to the X Server.

XUngrabServer restarts processing of requests and close-downs on other connections. If it is necessary to grab the X Server, do so only for short amounts of time because no processing of requests or closedowns on any connection occurs while the server is grabbed.

¦ Copyright IBM Corp. 1985, 1991

2.4.366 XUninstallColormap

XUninstallColormap(*display, cmap*)
Display **display*;
Colormap *cmap*;

display Specifies the connection to the X Server.

cmap Specifies the colormap ID.

XUninstallColormap removes the specified colormap from the required list for its screen. As a result, the specified *cmap* will be uninstalled, and the X Server might implicitly install or uninstall additional colormaps. Which colormaps get installed or uninstalled is server-dependent, but the required list must remain installed.

If the specified colormap becomes uninstalled, the X Server generates a **ColormapNotify** event on every window that has the *cmap* as the colormap resource ID. In addition, the X Server generates a **ColormapNotify** event on every window that has that *cmap* as the colormap resource ID if the colormap was uninstalled as a result of this function.

XUninstallColormap can generate the event error **BadColor**.

¦ Copyright IBM Corp. 1985, 1991

2.4.367 XUnionRectWithRegion

XUnionRectWithRegion(*rectangle, src_region, dest_region*)
Rectangle **rectangle*;
Region *src_region*
Region *dest_region*;

rectangle Specifies the rectangle.

src_region Specifies the source region to be used.

dest_region Specifies the destination region.

XUnionRectWithRegion creates the destination region from a union of the specified rectangle and the specified source region.

¦ Copyright IBM Corp. 1985, 1991

2.4.368 XUnionRegion

XUnionRegion(*sra, srb, dr*)
Region *sra, srb, dr*;

sra, srb Specifies the two regions for the computation.

dr Stores the result of the computation.

XUnionRegion computes the union of two regions.

¦ Copyright IBM Corp. 1985, 1991

2.4.369 XUniqueContext

XContext XUniqueContext()

XUniqueContext creates a unique context type that may be used in subsequent calls to **XSaveContext** and **XFindContext**.

¦ Copyright IBM Corp. 1985, 1991

2.4.370 XUnloadFont

XUnloadFont(*display, font*)
Display **display*;
Font *font*;

display Specifies the connection to the X Server.

font Specifies the font ID.

XUnloadFont unloads the specified font loaded by **XLoadFont**. It deletes the association between the font resource ID and the specified font. The font is freed when no other resource references it. The font can be unloaded if this is the last reference to it. In any case, the font should not be referenced again because the resource ID has been destroyed.

This routine can generate the event error **BadFont**.

¦ Copyright IBM Corp. 1985, 1991

2.4.371 XUnmapSubwindows

XUnmapSubwindows(*display, window*)
Display **display*;
Window *window*;

display Specifies the connection to the X Server.

window Specifies the window ID.

XUnmapSubwindows unmaps all subwindows for the specified window in bottom-to-top stacking order. The X Server generates an **UnmapNotify** event on each subwindow and exposure events on formerly obscured windows. This function allows the X Server to unmap all of the windows at one time. See "Mapping Windows" in topic 1.5.6.

XUnmapSubwindows can generate the event error **BadWindow**.

¦ Copyright IBM Corp. 1985, 1991

2.4.372 XUnmapWindow

XUnmapWindow(*display, window*)
Display **display*;
Window *window*;

display Specifies the connection to the X Server.

window Specifies the window ID.

XUnmapWindow unmaps the specified window. The X Server generates an **UnmapNotify** event. Unmapping a window generates exposure events on formerly obscured windows. See "Mapping Windows" in topic 1.5.6.

If the specified window is already unmapped, **XUnmapWindow** has no effect. Normal exposure processing on formerly obscured windows is performed. The subwindows are still mapped but are not visible until the parent window is mapped again.

XUnmapWindow can generate the event error **BadWindow**.

¦ Copyright IBM Corp. 1985, 1991

2.4.373 XUseKeymap

Status XUseKeymap(*keymapfile*)
char **keymapfile*;

keymapfile Specifies the name of the keymap file to use with the current process.

XUseKeymap provides an alternate keymap file for **XLookupMapping**. It changes the keymap file. This change only affects the keymap within the current process. It returns a zero if it cannot find the keymap file named **keymapfile** or if the file contains a bad magic number. If **XUseKeymap** fails, the existing keymap is untouched.

¦ Copyright IBM Corp. 1985, 1991

2.4.374 XVisualIDFromVisual

VisualID XVisualIDFromVisual(*visual*)
     Visual **visual*;

visual Specifies the visual type

XVisualIDFromVisual returns the visual ID for the specified visual type.

¦ Copyright IBM Corp. 1985, 1991

2.4.375 XWarpPointer

XWarpPointer(*display, src_w, dest_w, src_x, src_y,*
src_width, src_height, dest_x, dest_y)
Display **display*;
Window *src_w, dest_w*;
int *src_x, src_y*;
unsigned int *src_width, src_height*;
int *dest_x, dest_y*;

display Specifies the connection to the X Server.

src_w Specifies the window ID of the source window. It can be set to the window ID or the constant **None**.

dest_w Specifies the window ID of the destination window. It can be set to the window ID or the constant **None**.

src_x, src_y Specifies the *x* and *y* coordinates, which are relative to the origin of the source window, within the source window.

src_width, src_height Specifies the width and height of the source window.

dest_x, dest_y Specifies the *x* and *y* coordinates, which are relative to the origin of the destination window, within the destination window.

XWarpPointer moves the pointer to an arbitrary point on the screen. It moves the pointer to the location specified by the *dest_x* and *dest_y* arguments.

If the destination window is **None**, the pointer is moved by offsets specified by the *dest_x* and *dest_y* coordinates. The pointer should normally be left to the mouse.

XWarpPointer generates events just as moving the pointer from one position to another generates events. Do not use **XWarpPointer** to move the pointer outside the *confine_to* window of an active pointer grab. Otherwise, the pointer will be moved as far as the closest edge of the *confine_to* window.

If the *src_w* argument is **None**, the move is independent of the current pointer position. However, if *src_w* is a window, the move only takes place if the pointer is currently contained in a visible portion of the specified rectangle of the *src_w*.
If *dest_w* is **None**, the function moves the pointer by the offsets, *dest_x*, *dest_y* relative to the current position of the pointer.
If *dest_w* is a window, the function moves the pointer to the offsets *dest_x*, *dest_y* relative to the origin of *dest_w*. However, if *src_w* is not **None**, the move only takes place if the pointer is currently contained in a visible portion of the specified rectangle of the *src_w*.
If *src_height* is zero, the function replaces it with the current height of the source window minus *src_y*.
If *src_width* is zero, the function replaces it with the current width of the source window minus *src_x*.

XWarpPointer can generate the event error **BadWindow**.

¦ Copyright IBM Corp. 1985, 1991

2.4.376 XWindowEvent

XWindowEvent(*display, window, event_mask, event_return*)
Display **display*;
Window *window*;
long *event_mask*;
XEvent **event_return*;

display Specifies the connection to the X Server.

window Specifies the window ID.

event_mask Specifies the event mask which is the bitwise inclusive OR of one or more of the valid event mask bits. See "Defining Event Masks" in topic 1.11.3.

event_return Copies the matched event structure into this client-supplied structure.

XWindowEvent removes the next event that matches both a window and an event mask. It searches the event queue for this event. When **XWindowEvent** finds a match, it removes that event from the queue and copies it into the specified **XEvent** structure. (See "Defining Event Structures" in topic 1.11.2.) The other events stored in the queue are not discarded.

If the event requested is not in the queue, **XWindowEvent** flushes the output buffer and blocks until another event is received.

¦ Copyright IBM Corp. 1985, 1991

2.4.377 XWriteBitmapFile

int XWriteBitmapFile(*display, filename, bitmap, width, height, x_hot, y_hot*)
Display **display*;
char **filename*;
Pixmap *bitmap*;
int *width*, *height*;
int *x_hot*, *y_hot*;

display Specifies the connection to the X Server.

filename Specifies the filename to use. The format for this filename is operating system specific.

bitmap Specifies the *bitmap* to be written.

width, height Specifies the width and height of the bitmap to be written.

x_hot, y_hot Specifies the hotspot coordinates.

XWriteBitmapFile writes an X-Windows Version 2.1 bitmap to a file. This function only writes out X-Windows Version 2.1 format. Refer to the example program diskette for an example of a bitmap file.

If *x_hot* and *y_hot* are not **-1, -1**, **XWriteBitmapFile** writes them out as the hotspot coordinates for the bitmap. The location of *x_hot* and *y_hot* is **-1, -1** if not specified in the file.

XWriteBitmapFile returns:

BitmapOpenFailed, if the file cannot be opened for writing.
BitmapNoMemory, if insufficient memory is allocated.
BitmapSuccess, if it is successful.

¦ Copyright IBM Corp. 1985, 1991

2.4.378 XXorRegion

XXorRegion(*sra*, *srb*, *dr*)
Region *sra*, *srb*, *dr*;

sra, srb Specifies the two regions for the computation.

dr Stores the result of the computation.

XXorRegion calculates the difference between the union and the intersection of two regions.

¦ Copyright IBM Corp. 1985, 1991

2.4.379 Example of a C language Program

The following program is an example of an X-Windows application. This program may be useful if the contents of your screen have been corrupted by a program error (such as unintentionally using the **RootWindow**) or by messages put out by the system underneath your windows.

```
#include <stdio.h>
#include <X11/Xlib.h>
#include <X11/Xutil.h>

main(argc, argv)
     int     argc;
     char  **argv;
{
     Window window;
     Display *display;
     XWMHints wmhints;
     XSizeHints sizehints;

     if ((display = XOpenDisplay(argc ? argv[1]:"")) == NULL )
            fprintf(stderr, "Could not open Display");

     window = XCreateSimpleWindow(display, DefaultRootWindow(display), 0, 0,
                                  DisplayWidth(display, DefaultScreen(displa
                                  DisplayHeight(display, DefaultScreen(disp
                                  BlackPixel(display, DefaultScreen(display

     wmhints.flags = StateHint;
     wmhints.initial_state = NormalState;
     XSetWMHints(display, window, &wmhints);

     sizehints.flags = ( USPosition | USSize ) ;
     sizehints.x = 0 ;
     sizehints.y = 0 ;
     sizehints.width  = DisplayWidth(display, DefaultScreen(display));
     sizehints.height = DisplayHeight(display, DefaultScreen(display));
     XSetNormalHints(display, window, &sizehints );

     XMapWindow(display, window);             /* Put it on the screen   */
     XSync(display, 1) ;                          /*   Make sure it get
     XCloseDisplay(display);                    /* Disconnect from server

}
```

This program basically connects to the display, creates a window with a black background and zero-width border over the root window, maps the window to the screen, and then destroys it. Then, exposure events are sent to the client programs that have selected them on all mapped unobscured windows. This causes most clients to repaint their windows.

¦ Copyright IBM Corp. 1985, 1991

3.0 Chapter 3. FXlib Functions

Subtopics
3.1 CONTENTS
3.2 About This Chapter
3.3 Naming and Argument Conventions within FXlib
3.4 Programming Considerations
3.5 Subroutine and Function Reference Information

¦ Copyright IBM Corp. 1985, 1991

3.1 CONTENTS

¦ Copyright IBM Corp. 1985, 1991

3.2 About This Chapter

FORTRAN 77 X-Windows Library Functions and Subroutines

FXlib is a library of FORTRAN 77 bindings to the **Xlib** C language routine library. **FXlib** routines are C language routines called in the FORTRAN 77 program. The **FXlib** routine sets up and interfaces with the appropriate C language **Xlib** function. This chapter provides a list of these routines.

A basic understanding of a graphics window system and FORTRAN 77 is required.

The FORTRAN 77 X-Windows functions and subroutines are supported for the AIX RT X-Windows, Version 2.1 only.

¦ Copyright IBM Corp. 1985, 1991

3.3 Naming and Argument Conventions within FXlib

The following are the naming and syntax conventions for **FXlib** functions.

To differentiate **FXlib** routines from C language **Xlib** routines:

FXlib function names use all lowercase characters.

FXlib function names all begin with the lowercase characters fx;
Xlib function names all begin with the uppercase character X.

The argument list for an **FXlib** function is the same as the argument list for the equivalent function in **Xlib**.

In **FXlib**, the type of data structure is usually referenced as "type **data structure name**". For example, the "type **Screen**" refers to a data structure called **Screen**.

The data structures are described in more detail in the FORTRAN 7 include files found in directory **/usr/include/X11**.

¦ Copyright IBM Corp. 1985, 1991

3.4 Programming Considerations

The major programming considerations are:

The **FXlib** routine can be declared external by the FORTRAN 77 program.

The FORTRAN 77 program must declare the **FXlib** functions as **integer*4** when the function returns a value. FORTRAN 77 subroutines are not declared.

The FORTRAN 77 variable declarations used in the binding procedur calling sequence examples that follow are for illustrative purposes only. The asterisk (*) in the left column is used as a continuation character in the routine examples.

Predefined values for binding procedure parameters are given in th FORTRAN 77 include files, for example, **Xlib.f**.

FORTRAN 77 programs using X-Windows often need to access memory data areas created by the windowing system. The FORTRAN 77 interface to the X-Windows system provides:

A set of memory management function

A set of include files that provides access to specific data structur elements.

Generally, whenever a FORTRAN 77 X-Windows interface routine specifies an address as an input parameter or a data structure as a return parameter, the user needs to manipulate user memory to set parameters or retrieve data. Data structure elements can be accessed by a manual method using parameter table lookup, or by an automatic method using macro definitions.

Subtopics

¦ Copyright IBM Corp. 1985, 1991

3.4.1 Example Programs

There are examples of how to use each FORTRAN 77 function in directory **(/usr/lpp/X11/samples/fortran)**. It is recommended that you use these examples to help understand the interface between C language and FORTRAN 77.

Subtopics
3.4.1.1 Memory Management Routines

¦ Copyright IBM Corp. 1985, 1991

3.4.1.1 Memory Management Routines

The FORTRAN 77 X-Windows memory management routines provide a means to manipulate user memory and access specific data structure elements. For a more detailed explanation of these functions, see "Subroutine and Function Reference Information" in topic 3.5. The following is a summary:

fxallocatememory -- Allocate a segment of memory. The size of th segment is specified and the starting address of the segment is returned.

fxfreememory -- Free a segment of memory. The starting address of th segment is specified.

fxincrementaddress -- Increment an address. The number of bytes t increment the address is specified and the new address is returned.

fxgetvalue -- Get a 1, 2, or 4 byte integer value from memory. Th address of where the value is stored is determined from a specified starting address and a specified offset from the starting address. The integer value of the specified length is returned.

fxputvalue -- Put a 1, 2, or 4 byte integer value into memory. Th address of where to store the value is determined from a specified starting address and an offset from the starting address.

fxgetstring -- Get a character string value from memory. The addres of where the string is stored is contained in an intermediate address which is derived from a specified starting address and a specified offset from the starting address. The null-terminated string is returned.

fxputstring -- Put a character string value into memory. The addres of where the string is to be stored is contained in an intermediate address, which is determined from a specified starting address and a specified offset from the starting address. The string must be null-terminated.

fxgetstringaddress -- Get an address of a character string variable The string is specified and the address is returned as an integer value.

fxgetstringataddress -- Get a character string value from memory. Th address of where the string is stored is contained in the specified input variable. The null-terminated string is returned.

The memory management routines require that several parameters be specified. The starting address parameter is generally either the address of a FORTRAN 77 allocated data area or a return address from a FORTRAN 77 X library subroutine call. The offset and size parameters are provided in include files.

¦ Copyright IBM Corp. 1985, 1991

3.4.2 Include Files

The FORTRAN 77 X-Windows include files (in directory **/usr/include/X11**) provide a means for accessing particular data structure elements. An include file has a separate section for each type of data structure. Each data structure section is divided into a table and a list of macro definitions. The table is used to manually find parameter values for use in calls to the memory management functions. The macro definitions are used to automatically provide both the parameter values and the call to the appropriate memory management function.

The following is an example of the information contained in an include file:

```
_
_   Data structure for graphics example.
_
_
_  structure - XGE
_   memory allocation - 16 bytes
_
```

	member	**off**	**len**	**comment**
_	----	---	---	-----
_	function	0	4	logical operation
_	*plane_mask*	4	2	*plane_mask*
_	foreground	6	1	foreground pixel
_	background	7	1	background pixel
_	name	8	4	name (string)
_	*line_style*	12	4	**LineSolid, LineOnOffDash, LineDoubleDash**

```
_
define( XGE_memory_size, 16) dnl
_
define( allocate_XGE, fxallocatememory (16) ) dnl
define( free_XGE, fxfreememory ($1) ) dnl
_
define( get_XGE_function, fxgetvalue($1,   0, 4) ) dnl
define( get_XGE_plane_mask, fxgetvalue($1,   4, 2) ) dnl
define( get_XGE_foreground, fxgetvalue($1,   6, 1) ) dnl
define( get_XGE_background, fxgetvalue($1,   7, 1) ) dnl
define( get_XGE_name, fxgetstring($1,   8) ) dnl
define( get_XGE_line_style, fxgetvalue($1,  12, 4) ) dnl
_
define( put_XGE_function,   fxputvalue($1,   0, 4, $2) ) dnl
define( put_XGE_plane_mask, fxputvalue($1,   4, 2, $2) ) dnl
define( put_XGE_foreground, fxputvalue($1,   6, 1, $2) ) dnl
define( put_XGE_background, fxputvalue($1,   7, 1, $2) ) dnl
define( put_XGE_name,fxputstring($1,   8, $2) ) dnl
define( put_XGE_line_style, fxputvalue($1,  12, 4, $2) ) dnl
```

¦ Copyright IBM Corp. 1985, 1991

3.4.3 Manual Table Usage

The table section in the include files provides a means to manually access the parameters required by the memory management routines in order to access a particular data structure element. When using the table, it is not necessary to place the include file in the FORTRAN 77 source file, and it is not necessary to preprocess the source file before compilation.

The terms used in the lookup table are defined as follows:

structure The structure name used by the windowing system.

memory allocation The size of the contiguous memory area for the structure.

member The name of the data element.

off The memory offset in bytes of the element from the start of the memory area.

len The length in bytes of the element.

comment A description of the element.

The table lookup section is used directly with the memory management functions as follows:

Allocating and freeing memory

The memory allocation parameter in the table is used to allocate memory. No table lookup is needed to free memory. From the example include file on page 3.4.2, the memory management routines to allocate and free memory are:

```
integer*4 startaddress
startaddress = fxallocatememory (16)
fxfreememory (startaddress)
```

Storing and retrieving integer data

The off and len parameters in the table are used to store and retrieve integer data. From the example include file on page 3.4.2, the memory management routines to store and retrieve integer data for member *plane_mask* are:

```
integer*4 integervalue
integervalue = fxgetvalue (startaddress, 4, 2)
fxputvalue (startaddress, 4, 2, integervalue)          .
```

Storing and retrieving string data

The off parameter in the table is used to store and retrieve string data. From the example include file on page 3.4.2, the memory management routines to store and retrieve string data for member name are:

```
character*12 stringvalue
stringvalue = fxgetstring (startaddress, 8)
fxputstring (startaddress, 8, stringvalue)
```

¦ Copyright IBM Corp. 1985, 1991

3.4.4 Automatic Macro Definition Usage

The macro definitions in an include file provide an automatic means of accessing both the parameters and the memory management routine to access a particular data structure element. The macro definitions correspond to the format used by the **m4** macro processor.

The **m4** processor must be invoked on any source file that uses a FORTRAN 77 include file prior to compilation. For example, if the FORTRAN 77 source file **input.f** uses a FORTRAN 77 include file, the preprocessor could be invoked as follows:

```
m4 -Bnum input.f > output.f
```

Output file **output.f** could then be used as input to the FORTRAN 77 compiler. The **m4** option **-Bnum** is only required if the **m4** processor needs additional work space.

An include file is invoked by putting an include statement at the beginning of the FORTRAN 77 source file. For example, to use the FORTRAN 77 include file **Xlib.f** in directory **/usr/include/X11**, the following statement is put in the source file beginning at column 1:

```
include( /usr/include/X11/Xlib.f ) dnl
```

The include file name must be the full path name. The **m4** macro **dnl** is optional, and suppresses extra lines being inserted into the source file.

The macro definitions are formed and invoked as follows:

Allocating and freeing memory

The macro name for allocating memory is formed by concatenating the string **allocate_** and the structure name. The macro name for freeing memory is formed by concatenating the string **free_** and the structure name. The macro for allocating memory has no parameters. The macro for freeing memory has the starting address of the data area to be freed as a parameter. From the example include file on page 3.4.2, the macro definitions to allocate and free data are invoked by:

```
integer*4 startaddress
startaddress = allocate_XGE
free_XGE (startaddress)
```

Storing and retrieving data

The macro name for storing and retrieving both integer and string data from memory is formed by concatenating either the string **get_** (to retrieve) or **put_** (to store), the structure name and the data element name. From the example include file on page 3.4.2, the macro definitions to store and retrieve integer and string data are invoked by:

```
integer*4 planemask
planemask = 7
put_XGE_planemask (startaddress, planemask)
planemask = get_XGE_planemask (startaddress)

character*12 name
name = "example\0"
put_XGE_name (startaddress, name)
name = get_XGE_name (startaddress)
```

¦ Copyright IBM Corp. 1985, 1991

3.4.5 User Guidelines

The following guidelines should be observed when using the macro definitions in the include files and the **m4** processor:

All macro definitions must begin in coloumn 1

For macro definitions with arguments, no white space is allowe between the macro name and the left parenthesis.

If the word unix is used in the FORTRAN 77 source file (unix:0 fo example), and the **m4** processor is to be invoked, the following macro must be placed at the beginning of the file:

```
undefine( 'unix' ) dnl
```

For convenience, this statement has been put in each FORTRAN 77 include file. If a FORTRAN 77 include file is used in the source file, this statement is automatically placed in the file.

Lines in an include file (and consequently a source file in which th include file is used) that begin with an underscore character are treated as comments.

¦ Copyright IBM Corp. 1985, 1991

3.5 Subroutine and Function Reference Information

The rest of this chapter contains more detailed information about FORTRAN 77 subroutines and functions. The routines are arranged in alphabetical order. The FORTRAN 77 routines are in the **libXfx.a** file.

Some FORTRAN 77 subroutines generate error codes that are documented in Appendix F, "Error Codes" in topic F.0.

Subtopics

¦ Copyright IBM Corp. 1985, 1991

¦ Copyright IBM Corp. 1985, 1991

¦ Copyright IBM Corp. 1985, 1991

¦ Copyright IBM Corp. 1985, 1991

¦ Copyright IBM Corp. 1985, 1991

¦ Copyright IBM Corp. 1985, 1991

¦ Copyright IBM Corp. 1985, 1991

3.5.1 fxactivatescreensaver

external fxactivatescreensaver
integer*4 *display*

call fxactivatescreensaver(*display*)

display Specifies the connection to the X Server.

fxactivatescreensaver activates the screen saver.

For more information about the C language **Xlib** routine called by the **FXlib** binding routine **fxactivatescreensaver**, see the **XActivateScreenSaver** routine in "Xlib Functions."

¦ Copyright IBM Corp. 1985, 1991

3.5.2 fxaddhost

external fxaddhost
integer*4 *display, host*

call fxaddhost(*display, host*)

display Specifies the connection to the X Server.

host Specifies the address of type **XHostAddress** of the network of the host machine to be added to the access control list.

fxaddhost adds a single host.

This routine can generate the errors **BadAlloc** and **BadValue**.

For more information about the C language **Xlib** routine called by the **FXlib** binding routine **fxaddhost**, see the **XAddHost** routine in "Xlib Functions."

¦ Copyright IBM Corp. 1985, 1991

3.5.3 fxaddhosts

external fxaddhosts
integer*4 *display, hosts, numhosts*

call fxaddhosts(*display, hosts, numhosts*)

display Specifies the connection to the X Server.

hosts Specifies the starting address of type **XHostAddress** for a list of each host that is to be added to the access control list. The client must allocate the host list and must free the host list when it is no longer needed.

numhosts Specifies the number of hosts.

fxaddhosts adds multiple hosts at one time.

This routine can generate the errors **BadAlloc** and **BadValue**.

For more information about the C language **Xlib** routine called by the **FXlib** binding routine **fxaddhosts**, see the **XAddHosts** routine in "Xlib Functions."

¦ Copyright IBM Corp. 1985, 1991

3.5.4 fxaddpixel

```
integer*4 fxaddpixel
external  fxaddpixel
integer*4 ximage, value
integer*4 status

status = fxaddpixel( ximage, value)
```

ximage Specifies the address of type **XImage** of the image structure.

value Specifies the constant *value* that is to be added.

fxaddpixel increments each pixel in the pixmap by a constant value.

For more information about the C language **Xlib** routine called by the **FXlib** binding routine **fxaddpixel**, see the **XAddPixel** routine in "Xlib Functions."

¦ Copyright IBM Corp. 1985, 1991

3.5.5 fxaddtosaveset

```
external  fxaddtosaveset
integer*4 display
integer*4 windowadd

call fxaddtosaveset( display, windowadd)
```

display Specifies the connection to the X Server.

windowadd Specifies the destination window ID of the children to be added to the client save-set.

fxaddtosaveset adds a window to the client's save-set.

This routine can generate the errors **BadWindow** and **BadMatch**.

For more information about the C language **Xlib** routine called by the **FXlib** binding routine **fxaddtosaveset**, see the **XAddToSaveSet** routine in "Xlib Functions."

¦ Copyright IBM Corp. 1985, 1991

3.5.6 fxallocatememory

```
integer*4 fxallocatememory
external fxallocatememory
integer*4 numberbytes
integer*4 startaddress

startaddress = fxallocatememory( numberbytes)
```

numberbytes Specifies the number of bytes to allocate for the desired memory area.

fxallocatememory allocates a segment of memory.

Return value *startaddress* contains the address of the allocated area.

¦ Copyright IBM Corp. 1985, 1991

3.5.7 fxalloccolor

integer*4 fxalloccolor
external fxalloccolor
integer*4 *display*
integer*4 *cmap*
integer*4 *screendefreturn*
integer*4 *status*

status = **fxalloccolor**(*display, cmap, screendefreturn*)

display Specifies the connection to the X Server.

cmap Specifies the colormap ID.

screendefreturn Returns an address of type **XColor** that has the values used in the colormap.

fxalloccolor obtains the closest color provided by the hardware.

The variable *status* contains the status of the color allocation. If this routine finds a problem, it returns a zero. If it is successful, it returns a nonzero.

fxalloccolor can generate the event errors **BadAlloc** and **BadColor**.

For more information about the C language **Xlib** routine called by the **FXlib** binding routine **fxalloccolor**, see the **XAllocColor** routine in "Xlib Functions."

¦ Copyright IBM Corp. 1985, 1991

3.5.8 fxalloccolorcells

```
integer*4 fxalloccolorcells
external  fxalloccolorcells
integer*4 display
integer*4 cmap
integer*4 contig, planemasksreturn
integer*4 nplanes, pixelsreturn, ncolors
integer*4 status

status = fxalloccolorcells( display, cmap, contig,
*                           planemasksreturn, nplanes, pixelsreturn, ncolors)
```

display	Specifies the connection to the X Server.
cmap	Specifies the colormap ID.
contig	Specifies a Boolean value.
planemasksreturn	Returns an array of plane masks.
nplanes	Specifies the number of pixel values that are to be returned in the planes mask array.
pixelsreturn	Returns an array of pixel values.
ncolors	Specifies the number of pixel values that are to be returned in the pixelsreturn array.

fxalloccolorcells allocates color cells.

The variable *contig* can be set to the value of 1 if the planes must be contiguous, or, it can be set to the value of zero if the planes do not need to be contiguous.

fxalloccolorcells can generate the errors **BadColor, BadValue,** and **BadAlloc**.

For more information about the C language **Xlib** routine called by the **FXlib** binding routine **fxalloccolorcells**, see the **XAllocColorCells** routine in "Xlib Functions."

¦ Copyright IBM Corp. 1985, 1991

3.5.9 fxalloccolorplanes

```
integer*4  fxalloccolorplanes
external  fxalloccolorplanes
integer*4 display
integer*4 cmap
integer*4 contig, pixelsreturn
integer*4 ncolors, nreds, ngreens, nblues
integer*4 rmaskreturn, gmaskreturn
integer*4 bmaskreturn
integer*4 status

status = fxalloccolorplanes( display, cmap, contig,
*                                pixelsreturn, ncolors, nreds, ngreens, nblue
*                                rmaskreturn, gmaskreturn, bmaskreturn)
```

display	Specifies the connection to the X Server.
cmap	Specifies the colormap ID.
contig	Specifies a Boolean value. This variable should be set to the value 1 if the planes must be contiguous; or, to the value zero if the planes do not need to be contiguous.
pixelsreturn	Returns an array of pixel values.
ncolors	Specifies the number of pixel values that are to be returned in the *pixelsreturn* array.
nreds	Specifies the number of red colors (shades). This value must be a non-negative value.
ngreens	Specifies the number of green colors (shades). This value must be a non-negative value.
nblues	Specifies the number of blue colors (shades). This value must be a non-negative value.
rmaskreturn	Returns bit masks for the red planes.
gmaskreturn	Returns bit masks for the green planes.
bmaskreturn	Returns bit masks for the blue planes.

fxalloccolorplanes allocates color planes.

fxalloccolorplanes can generate the errors **BadAlloc, BadColor,** and **BadValue**.

For more information about the C language **Xlib** routine called by the **FXlib** binding routine **fxalloccolorplanes**, see the **XAllocColorPlanes** routine in "Xlib Functions."

¦ Copyright IBM Corp. 1985, 1991

3.5.10 fxallocnamedcolor

```
integer*4 fxallocnamedcolor
external  fxallocnamedcolor
integer*4 display
integer*4 cmap
character*256  colorname
integer*4 screendefreturn
integer*4 exactdefreturn
integer*4 status

status = fxallocnamedcolor( display, cmap, colorname,
*                                   screendefreturn, exactdefreturn)
```

display	Specifies the connection to the X Server.
cmap	Specifies the colormap ID.
colorname	Specifies the color-name string for the color-definition structure returned, for example *red*.
screendefreturn	Returns an address of type **XColor** that has the values used in the colormap.
exactdefreturn	Returns an address of type **XColor** that has the true pixel values that indicate the closest color provided by the hardware for the specified color name.

fxallocnamedcolor obtains the color definition data structure for a specified color. It obtains the closest color supported by the hardware.

The variable *status* contains the status of the color allocation. If an error is encountered, it returns a zero. If it is successful, it returns a non-zero.

fxallocnamedcolor can generate the errors **BadAlloc, BadColor**, and **BadName**.

For more information about the C language **Xlib** routine called by the **FXlib** binding routine **fxallocnamedcolor**, see the **XAllocNamedColor** routine in "Xlib Functions."

¦ Copyright IBM Corp. 1985, 1991

3.5.11 fxallowevents

external fxallowevents
integer*4 *display*
integer*4 *eventmode, time*

call fxallowevents(*display, eventmode, time*)

display Specifies the connection to the X Server.

eventmode Specifies the event mode.

time Specifies the time in a time stamp, which is expressed in milliseconds, or **CurrentTime**.

fxallowevents allows further events to be processed when the device is frozen.

The variable *eventmode* can be set to one of the following:

AsyncPointer **SyncPointer**

AsyncKeyboard **SyncKeyboard**

ReplayPointer **ReplayKeyboard**

AsyncBoth **SyncBoth**.

This routine can generate the error **BadValue**.

For more information about the C language **Xlib** routine called by the **FXlib** binding routine **fxallowevents**, see the **XAllowEvents** routine in "Xlib Functions."

¦ Copyright IBM Corp. 1985, 1991

3.5.12 fxallplanes

```
integer*4  fxallplanes
external   fxallplanes
integer*4  rc

rc = fxallplanes()
```

fxallplanes obtains a value that selects all planes of the display.

Return value *rc* is a value with all bits set on, suitable for use in a *planes* argument to a procedure.

For more information about the C language **Xlib** routine called by the **FXlib** binding routine **fxallplanes**, see the **XAllPlanes** routine in "Xlib Functions."

¦ Copyright IBM Corp. 1985, 1991

3.5.13 fxautorepeatoff

external fxautorepeatoff
integer*4 *display*

call fxautorepeatoff(*display*)

display Specifies the connection to the X Server.

fxautorepeatoff turns off the keyboard auto-repeat.

For more information about the C language **Xlib** routine called by the **FXlib** binding routine **fxautorepeatoff**, see the **XAutoRepeatOff** routine in "Xlib Functions."

¦ Copyright IBM Corp. 1985, 1991

3.5.14 fxautorepeaton

external fxautorepeaton
integer*4 *display*

call fxautorepeaton(*display*)

display Specifies the connection to the X Server.

fxautorepeaton turns on the keyboard auto-repeat.

For more information about the C language **Xlib** routine called by the **FXlib** binding routine **fxautorepeaton**, see the **XAutoRepeatOn** routine in "Xlib Functions."

¦ Copyright IBM Corp. 1985, 1991

3.5.15 fxbell

external fxbell
integer*4 *display, percent*

call fxbell(*display, percent*)

display Specifies the connection to the X Server.

percent Specifies the base volume for the bell, which can range from -100 to 100 inclusive.

fxbell rings the bell.

For more information about the C language **Xlib** routine called by the **FXlib** binding routine **fxbell**, see the **Xbell** routine in "Xlib Functions."

¦ Copyright IBM Corp. 1985, 1991

3.5.16 fxbitmapbitorder

```
integer*4  fxbitmapbitorder
external   fxbitmapbitorder
integer*4  display
integer*4  rc

rc = fxbitmapbitorder( display)
```

display Specifies the connection to the X Server.

fxbitmapbitorder returns the bitmap bit order.

Return value *rc* is the bitmap bit order.

For more information about the C language **Xlib** routine called by the **FXlib** binding routine **fxbitmapbitorder**, see the **XBitmapBitOrder** routine in "Xlib Functions."

¦ Copyright IBM Corp. 1985, 1991

3.5.17 fxbitmappad

```
integer*4  fxbitmappad
external   fxbitmappad
integer*4  display
integer*4  rc

rc = fxbitmappad( display)
```

display Specifies the connection to the X Server.

fxbitmappad returns the scanline padding.

Return value *rc* is the scanline padding bit value. Each scanline must be padded (bits) to some multiple of this value.

For more information about the C language **Xlib** routine called by the **FXlib** binding routine **fxbitmappad**, see the **XBitmapPad** routine in "Xlib Functions."

¦ Copyright IBM Corp. 1985, 1991

3.5.18 fxbitmapunit

```
integer*4  fxbitmapunit
external   fxbitmapunit
integer*4  display
integer*4  rc

rc = fxbitmapunit( display)
```

display Specifies the connection to the X Server.

fxbitmapunit returns the size of a bitmap's unit in bits.

Return value *rc* is the size of the bitmap's unit in bits. The scanline is calculated in multiples of this value.

For more information about the C language **Xlib** routine called by the **FXlib** binding routine **fxbitmapunit**, see the **XBitmapUnit** routine in "Xlib Functions."

¦ Copyright IBM Corp. 1985, 1991

3.5.19 fxblackpixel

```
integer*4  fxblackpixel
external   fxblackpixel
integer*4  display
integer*4  screen
integer*4  rc

rc = fxblackpixel( display, screen)
```

display Specifies the connection to the X Server.

screen Specifies the screen.

fxblackpixel obtains the black pixel for the screen specified.

Return value *rc* is the black pixel value for the specified screen.

For more information about the C language **Xlib** routine called by the **FXlib** binding routine **fxblackpixel**, see the **XBlackPixel** routine in "Xlib Functions."

¦ Copyright IBM Corp. 1985, 1991

3.5.20 fxblackpixelofscreen

```
integer*4  fxblackpixelofscreen
external   fxblackpixelofscreen
integer*4  screen
integer*4  rc

rc = fxblackpixelofscreen( screen)
```

screen Specifies the address of type **Screen** of the screen of the display.

fxblackpixelofscreen returns the black pixel value of the screen specified.

Return value *rc* is the black pixel value of the specified screen.

For more information about the C language **Xlib** routine called by the **FXlib** binding routine **fxblackpixelofscreen**, see the **XBlackPixelOfScreen** routine in "Xlib Functions."

¦ Copyright IBM Corp. 1985, 1991

3.5.21 fxcellsofscreen

```
integer*4  fxcellsofscreen
external   fxcellsofscreen
integer*4  screen
integer*4  rc

rc = fxcellsofscreen( screen)
```

screen Specifies the address of type **Screen** of the screen of the display.

fxcellsofscreen returns the number of colormap cells of the screen specified.

Return value *rc* is the number of colormap cells of the screen specified.

For more information about the C language **Xlib** routine called by the **FXlib** binding routine **fxcellsofscreen**, see the **XCellsOfScreen** routine in "Xlib Functions."

¦ Copyright IBM Corp. 1985, 1991

3.5.22 fxchangeactivepointergrab

external fxchangeactivepointergrab
integer*4 *display*
integer*4 *eventmask, cursor, time*

call fxchangeactivepointergrab(*display, eventmask, cursor,*
*** *time*)

display Specifies the connection to the X Server.

eventmask Specifies which pointer events are reported to the client. This is a bitwise inclusive OR of valid pointer event mask bits.

cursor Specifies the cursor that is displayed. This variable can be set to **None**.

time Specifies the time in a time stamp, which is expressed in milliseconds, or **CurrentTime**.

fxchangeactivepointergrab changes the active pointer.

The variable *eventmask* can be one of the following valid pointer event mask bits:

ButtonPressMask **ButtonReleaseMask**

EnterWindowMask **LeaveWindowMask**

PointerMotionMask **PointerMotionHintMask**

Button1MotionMask **Button2MotionMask**

Button3MotionMask **Button4MotionMask**

Button5MotionMask **ButtonMotionMask**

KeymapStateMask.

This routine can generate the error **BadCursor**.

For more information about the C language **Xlib** routine called by the **FXlib** binding routine **fxchangeactivepointer**, see the **XChangeActivePointer** routine in "Xlib Functions."

¦ Copyright IBM Corp. 1985, 1991

3.5.23 fxchangegc

external fxchangegc
integer*4 *display*
integer*4 *gc*
integer*4 *valuemaskchange*
integer*4 *values*

call fxchangegc(*display, gc, valuemaskchange, values*)

display	Specifies the connection to the X Server.
gc	Specifies the address of type **GC** that has the graphics context.
valuemaskchange	Specifies which components in the graphics context are to be changed using information in the **XGCValues** data structure. This argument is the bitwise inclusive OR of one or more of the valid **GC** component masks.
values	Specifies the address of the **XGCValues** data structure.

fxchangegc changes the components in the specified graphics context.

This routine can generate the errors **BadGC**, **BadAlloc**, **BadPixmap**, **BadFont**, **BadMatch**, and **BadValue**.

For more information about the C language **Xlib** routine called by the **FXlib** binding routine **fxchangegc**, see the **XChangeGC** routine in "Xlib Functions."

¦ Copyright IBM Corp. 1985, 1991

3.5.24 fxchangekeyboardcontrol

external fxchangekeyboardcontrol
integer*4 *display*
integer*4 *valuemask, values*

call fxchangekeyboardcontrol(*display, valuemask, values*)

display Specifies the connection to the X Server.

valuemask Specifies one value for each one bit in the mask. The values are in a least-to-most-significant bit order. The values are associated with the set of keys for the previously specified keyboard.

values Specifies the address of type **XKeyboardControl**.

fxchangekeyboardcontrol changes control from a keyboard.

This routine can generate the errors **BadMatch** and **BadValue**.

For more information about the C language **Xlib** routine called by the **FXlib** binding routine **fxchangekeyboardcontrol**, see the **XChangeKeyboardControl** routine in "Xlib Functions."

¦ Copyright IBM Corp. 1985, 1991

3.5.25 fxchangekeyboardmapping

```
external fxchangekeyboardmapping
integer*4 display
integer*4 firstkeycode
integer*4 keysymsperkeycode
integer*4 keysyms, numcodes

call fxchangekeyboardmapping( display, firstkeycode,
*                                    keysymsperkeycode, keysyms, numcodes
```

display	Specifies the connection to the X Server.
firstkeycode	Specifies the first keycode that is to be changed.
keysymsperkeycode	Specifies the *keysyms* that are to be used.
keysyms	Specifies the starting address of a list of keyboard symbols.
numcodes	Specifies the number of keycodes that are to be changed.

fxchangekeyboardmapping changes the keyboard mapping.

This routine can generate the errors **BadValue** and **BadAlloc**.

For more information about the C language **Xlib** routine called by the **FXlib** binding routine **fxchangekeyboardmapping**, see the **XChangeKeyboardMapping** routine in "Xlib Functions."

¦ Copyright IBM Corp. 1985, 1991

3.5.26 fxchangepointercontrol

```
external fxchangepointercontrol
integer*4 display
integer*4 doaccel, dothreshold
integer*4 accelnumerator, acceldenominator, threshold

call fxchangepointercontrol( display, doaccel,  dothreshold,
*                                 accelnumerator, acceldenominator,
*                                 threshold)
```

display	Specifies the connection to the X Server.
doaccel	Specifies a Boolean value that controls whether the values for the *accelnumerator* or *acceldenominator* are set.
dothreshold	Specifies a Boolean value that controls whether the value for the *threshold* is set.
accelnumerator	Specifies the numerator for the acceleration multiplier.
acceldenominator	Specifies the denominator for the acceleration multiplier.
threshold	Specifies the acceleration *threshold*.

fxchangepointercontrol controls the interactive feel of the pointer device.

The variables *doaccel* and *accelnumerator* can be set to **TRUE** or **FALSE**.

This routine can generate the error **BadValue**.

For more information about the C language **Xlib** routine called by the **FXlib** binding routine **fxchangepointercontrol**, see the **XChangePointerControl** routine in "Xlib Functions."

¦ Copyright IBM Corp. 1985, 1991

3.5.27 fxchangeproperty

```
external  fxchangeproperty
integer*4 display
integer*4 window
integer*4 property, type, format, mode
integer*4 data
integer*4 nelements

call fxchangeproperty( display, window, property, type, format,
*                              mode, data, nelements)
```

display Specifies the connection to the X Server.

window Specifies window ID for the window whose property is to be changed.

property Specifies the property atom.

type Specifies the type of the property.

format Specifies whether the data should be viewed as a list of 8-bit, 16-bit, or 32-bit quantities.

mode Specifies the mode of the operation.

data Specifies the address of the property data.

nelements Specifies the number of elements of the specified data format (8-bit, 16-bit, or 32-bit).

fxchangeproperty changes the property for a specified window. The property becomes undefined by one of the following:

The application calls **fxdeleteproperty**.

The application destroys the specified window

The application closes the last connection to the X Server

The X Server does not interpret the *type* of property, but passes *type* back to an application that later calls **fxgetproperty**.

The variable *mode* can be **PropModeReplace**, **PropModePrepend**, or **PropModeAppend**.

This routine can generate errors **BadWindow**, **BadAtom**, **BadAlloc**, or **BadValue**.

For more information about the C language **Xlib** routine called by the **FXlib** binding routine **fxchangeproperty**, see the **XChangeProperty** routine in "Xlib Functions."

¦ Copyright IBM Corp. 1985, 1991

3.5.28 fxchangesaveset

external fxchangesaveset
integer*4 *display*
integer*4 *window, changemode*

call fxchangesaveset(*display, window, changemode*)

display Specifies the connection to the X Server.

window Specifies the window ID.

changemode Specifies the mode.

fxchangesaveset adds or removes a window from the client save-set.

The *changemode* variable can be set to **SetModeInsert** or **SetModeDelete**. If *changemode* is **SetModeInsert**, **fxchangesaveset** adds the window to the client save-set. If *changemode* is **SetModeDelete**, **fxchangesaveset** deletes the window from the client save-set.

This routine can generate the errors **BadWindow**, **BadMatch**, and **BadValue**.

For more information about the C language **Xlib** routine called by the **FXlib** binding routine **fxchangesaveset**, see the **XChangeSaveSet** routine in "Xlib Functions."

¦ Copyright IBM Corp. 1985, 1991

3.5.29 fxchangewindowattributes

external fxchangewindowattributes
integer*4 *display*
integer*4 *window*
integer*4 *valuemask*
integer*4 *attribs*

call fxchangewindowattributes(*display, window, valuemask, attribs*)

display Specifies the connection to the X Server.

window Specifies window ID.

valuemask Specifies which window attributes are defined in the attribute argument. This mask is the bitwise inclusive OR of the valid attribute mask bits. If *valuemask* is zero, the rest is ignored, and the attributes are not referenced.

attribs Specifies the address of type **XSetWindowAttributes** that has the attributes of the window to change. The *valuemask* should have the appropriate bits set to indicate which attributes have been set in the data structure.

fxchangewindowattributes changes one or more window attributes.

For more information about the C language **Xlib** routine called by the **FXlib** binding routine **fxchangewindowattributes**, see the **XChangeWindowAttributes** routine in "Xlib Functions."

¦ Copyright IBM Corp. 1985, 1991

3.5.30 fxcheckifevent

```
integer*4 fxcheckifevent
external  fxcheckifevent
integer*4 display
integer*4 eventreturn
integer*4 predicate
character*256 arg
integer*4 rc

rc = fxcheckifevent( display, eventreturn, predicate, arg)
```

display Specifies the connection to the X Server.

eventreturn Specifies the destination address of **XEvent** in the client data area to copy the event structure from the event queue. The client must allocate the data area. The client must also free the data area when no longer needed.

predicate Specifies the procedure to call to determine if the next event in the queue matches the one specified by the event argument. The procedure must be declared **external**. Only the procedure name is passed to the binding routine.

arg Specifies the address of the user-supplied argument to pass to the predicate procedure.

fxcheckifevent flushes the output buffer and checks the event queue for the event specified without blocking.

Return value *rc* indicates the status of the specified event in the queue.

For more information about the C language **Xlib** routine called by the **FXlib** binding routine **fxcheckifevent**, see the **XCheckIfEvent** routine in "Xlib Functions."

fxcheckifevent requires a predicate procedure to pass as a parameter. The procedure determines if the event matches the one specified in the corresponding function. The predicate procedure is defined as follows:

```
external  functionname
integer*4 display
integer*4 event
integer*4 args

call functionname( display, event, args)
```

functionname Specifies the user-supplied function name.

display Specifies the connection to the X Server.

event Specifies the address of type **XEvent** in the client data area.

args Specifies the arguments passed from the **fxcheckifevent** function.

The *predicate* procedure is called once for each event in the queue until it finds a match between the event in the queue and the event specified by the corresponding function. After finding a match, the predicate

¦ Copyright IBM Corp. 1985, 1991

procedure returns **TRUE**. If it does not find a match, it returns **FALSE**.

¦ Copyright IBM Corp. 1985, 1991

3.5.31 fxcheckmaskevent

integer*4 fxcheckmaskevent
external fxcheckmaskevent
integer*4 *display*
integer*4 *eventmask, eventreturn*
integer*4 *rc*

rc = **fxcheckmaskevent**(*display, eventmask, eventreturn*)

display Specifies the connection to the X Server.

eventmask Specifies the event mask. This mask is the bitwise inclusive OR of one or more of the valid event mask bits.

eventreturn Specifies the destination address of type **XEvent** in the client data area to copy the event structure from the event queue. The client must allocate the data area. The client must also free the data area when it is no longer needed.

fxcheckmaskevent removes the next event that matches the mask that was passed. **XMaskEvent** returns a value of zero or a value of one if the event was returned.

Return value *rc* indicates the status of the specified event in the queue.

For more information about the C language **Xlib** routine called by the **FXlib** binding routine **fxcheckmaskevent**, see the **XCheckMaskEvent** routine in "Xlib Functions."

¦ Copyright IBM Corp. 1985, 1991

3.5.32 fxchecktypedevent

```
integer*4 fxchecktypedevent
external  fxchecktypedevent
integer*4 display
integer*4 eventtype, eventreturn

integer*4 rc
rc = fxchecktypedevent( display, eventtype, eventreturn)
```

display Specifies the connection to the X Server.

eventtype Specifies the event type to compare.

eventreturn Specifies the destination address of type **XEvent** in the client data area to copy the event structure from the event queue. The client must allocate the data area. The client must also free the data area when it is no longer needed.

fxchecktypedevent returns the next event in the queue that matches an event type.

Return value *rc* indicates the status of finding the specified event in the queue.

For more information about the C language **Xlib** routine called by the **FXlib** binding routine **fxchecktypedevent**, see the **XCheckTypedEvent** routine in "Xlib Functions."

¦ Copyright IBM Corp. 1985, 1991

3.5.33 fxchecktypedwindowevent

```
integer*4 fxchecktypedwindowevent
external  fxchecktypedwindowevent
integer*4 display
integer*4 window, eventtype, eventreturn
integer*4 rc

rc = fxchecktypedwindowevent( display, window, eventtype,
*                                          eventreturn)
```

display Specifies the connection to the X Server.

window Specifies the window ID.

eventtype Specifies the event type to compare.

eventreturn Specifies the destination address of type **XEvent** in the client data area to copy the event structure from the event queue. The client must allocate the data area. The client must also free the data area when it is no longer needed.

fxchecktypedwindowevent returns the next matched event in the queue for the specified window.

Return value *rc* indicates the status of finding the specified event in the queue.

For more information about the C language **Xlib** routine called by the **FXlib** binding routine **fxchecktypedwindowevent**, see the **XCheckTypedWindowEvent** routine in "Xlib Functions."

¦ Copyright IBM Corp. 1985, 1991

3.5.34 fxcheckwindowevent

```
integer*4 fxcheckwindowevent
external  fxcheckwindowevent
integer*4 display, window
integer*4 eventmask, eventreturn
integer*4 rc

rc = fxcheckwindowevent( display, window, eventmask, eventreturn)
```

display Specifies the connection to the X Server.

window Specifies the window ID.

eventmask Specifies the event mask. This mask is the bitwise inclusive OR of one or more of the valid event mask bits.

eventreturn Specifies the destination address of type **XEvent** in the client data area to copy the event structure from the event queue. The client must allocate the data area. The client must also free the data area when it is no longer needed.

fxcheckwindowevent removes the next event that matches both the window and the mask passed. It returns a value of zero or one.

Return value *rc* indicates the status of finding the specified event in the queue.

For more information about the C language **Xlib** routine called by the **FXlib** binding routine **fxcheckwindowevent**, see the **XCheckWindowEvent** routine in "Xlib Functions."

¦ Copyright IBM Corp. 1985, 1991

3.5.35 fxcirculatesubwindows

```
external  fxcirculatesubwindows
integer*4 display
integer*4 window
integer*4 direction

call fxcirculatesubwindows( display, window, direction)
```

display Specifies the connection to the X Server.

window Specifies window ID.

direction Specifies the direction for circulating the windows, up or down.

fxcirculatesubwindows circulates a subwindow up or down. Use **RaiseLowest** or **LowerHighest** to circulate the subwindows.

For more information about the C language **Xlib** routine called by the **FXlib** binding routine **fxcirculatesubwindows**, see the **XCirculateSubwindows** routine in "Xlib Functions."

¦ Copyright IBM Corp. 1985, 1991

3.5.36 fxcirculatesubwindowsdown

```
external  fxcirculatesubwindowsdown
integer*4 display
integer*4 window

call fxcirculatesubwindowsdown( display, window)
```

display Specifies the connection to the X Server.

window Specifies window ID.

fxcirculatesubwindowsdown lowers the highest mapped child of a window that partially or completely hides another child.

For more information about the C language **Xlib** routine called by the **FXlib** binding routine **fxcirculatesubwindowsdown**, see the **XCirculateSubwindowsDown** routine in "Xlib Functions."

¦ Copyright IBM Corp. 1985, 1991

3.5.37 fxcirculatesubwindowsup

```
external  fxcirculatesubwindowsup
integer*4 display
integer*4 window

call fxcirculatesubwindowsup( display, window)
```

display Specifies the connection to the X Server.

window Specifies window ID.

fxcirculatesubwindowsup raises the lowest mapped child of a hidden window.

For more information about the C language **Xlib** routine called by the **FXlib** binding routine **fxcirculatesubwindowsup**, see the **XCirculateSubwindowsUp** routine in "Xlib Functions."

¦ Copyright IBM Corp. 1985, 1991

3.5.38 fxcleararea

external fxcleararea
integer*4 *display*
integer*4 *window*
integer*4 *x, y*
integer*4 *width, height*
integer*4 *exposures*

call fxcleararea(*display, window, x, y, width, height, exposures*)

display Specifies the connection to the X Server.

window specifies the window ID.

x, y Specify the *x* and *y* coordinates which are relative to the origin of the window. These coordinates are the upper-left corner of the rectangle.

width, height Specify the *width* and *height* that are the dimensions of the rectangle to be cleared.

exposures Specifies a Boolean value of **TRUE** or **FALSE.**

fxcleararea clears a specified rectangular area in the specified window.

This routine can generate errors **BadWindow**, **BadMatch**, and **BadValue**.

For more information regarding the C language **Xlib** routine called by the **FXlib** binding routine **fxcleararea**, see the **XClearArea** routine in "Xlib Functions."

¦ Copyright IBM Corp. 1985, 1991

3.5.39 fxclearwindow

```
external  fxclearwindow
integer*4 display
integer*4 window

call fxclearwindow( display, window)
```

display Specifies the connection to the X Server.

window Specifies the window ID.

fxclearwindow clears the entire area in the specified window.

This routine can generate the errors **BadWindow**, **BadMatch**, and **BadValue**.

For more information about the C language **Xlib** routine called by the **FXlib** binding routine **fxclearwindow**, see the **XClearWindow** routine in "Xlib Functions."

¦ Copyright IBM Corp. 1985, 1991

3.5.40 fxclipbox

external fxclipbox
integer*4 *r, rect*

call fxclipbox(*r, rect*)

r Specifies the address of type **_XRegion** of the region structure. This is the region where the rectangle is located.

rect Specifies the address of type **XRectangle** of the rectangle structure. This is the rectangle where the smallest enclosing rectangle is generated.

fxclipbox generates the smallest enclosing rectangle in the rectangle.

For more information about the C language **Xlib** routine called by the **FXlib** binding routine **fxclipbox**, see the **XClipBox** routine in "Xlib Functions."

¦ Copyright IBM Corp. 1985, 1991

3.5.41 fxclosedisplay

external **fxclosedisplay**
integer*4 *display*

call fxclosedisplay(*display*)

display Specifies the connection to the X Server.

fxclosedisplay closes or disconnects a display from the X Server.

For more information about the C language **Xlib** routine called by the **FXlib** binding routine **fxclosedisplay**, see the **XCloseDisplay** routine in "Xlib Functions."

¦ Copyright IBM Corp. 1985, 1991

3.5.42 fxconfigurewindow

external fxconfigurewindow
integer*4 *display*
integer*4 *window*
integer*4 *valuemask*
integer*4 *values*

call fxconfigurewindow(*display, window, valuemask, values*)

display Specifies the connection to the X Server.

window Specifies window ID.

valuemask Specifies values that are to be set using information in the values data structure. This mask is a bitwise inclusive OR of valid change window values bits.

values Specifies the address of **XWindowChanges**, where the change values have been set.

fxconfigurewindow reconfigures the size, position, border, and stacking order of the window.

For more information about the C language **Xlib** routine called by the **FXlib** binding routine **fxconfigurewindow**, see the **XConfigureWindow** routine in "Xlib Functions."

¦ Copyright IBM Corp. 1985, 1991

3.5.43 fxconnectionnumber

```
integer*4  fxconnectionnumber
external   fxconnectionnumber
integer*4  display
integer*4  rc

rc = fxconnectionnumber( display)
```

display Specifies the connection to the X Server.

fxconnectionnumber obtains the connection number for the display specified.

Return value *rc* is the connection number (the file descriptor of the connection) for the specified display.

For more information about the C language **Xlib** routine called by the **FXlib** binding routine **fxconnectionnumber**, see the **XConnectionNumber** routine in "Xlib Functions."

¦ Copyright IBM Corp. 1985, 1991

3.5.44 fxconvertselection

external fxconvertselection
integer*4 *display*
integer*4 *selection, target*
integer*4 *property, requestor, time*

call fxconvertselection(*display, selection, target, property,*
** requestor, time*)

display Specifies the connection to the X Server.

selection Specifies the selection atom.

target Specifies the target atom.

property Specifies the property atom. This value can be set to **None**.

requestor Specifies the requestor.

time Specifies the time in either a timestamp, which is expressed in milliseconds, or **CurrentTime**.

fxconvertselection requests a selection conversion.

This routine can generate the errors **BadWindow** and **BadAtom**.

For more information about the C language **Xlib** routine called by the **FXlib** binding routine **fxconvertselection**, see the **XConvertSelection** routine in "Xlib Functions."

¦ Copyright IBM Corp. 1985, 1991

3.5.45 fxcopyarea

external fxcopyarea
integer*4 *display*
integer*4 *src, dest, gc*
integer*4 *srcx, srcy, width, height*
integer*4 *destx, desty*

call fxcopyarea(*display, src, dest, gc, srcx, srcy,*
** width, height, destx, desty*)

display Specifies the connection to the X Server.

src Specifies the source rectangle.

dest Specifies the destination rectangle.

srcx, srcy Specify the *x* and *y* coordinates of the source rectangle relative to its origin. These coordinates specify the upper-left corner of the source rectangle.

width Specifies the *width* of the source and destination rectangles.

height Specifies the *height* of the source and destination rectangles.

destx, desty Specify the *x* and *y* coordinates of the destination rectangle relative to its origin. These coordinates specify the upper-left corner of the source rectangle.

fxcopyarea copies an area of the specified drawable between drawables of the same root and depth.

This routine can generate the errors **BadDrawable**, **BadMatch**, and **BadGC**.

For more information about the C language **Xlib** routine called by the **FXlib** binding routine **fxcopyarea**, see the **XCopyArea** routine in "Xlib Functions."

¦ Copyright IBM Corp. 1985, 1991

3.5.46 fxcopycolormapandfree

```
integer*4 fxcopycolormapandfree
external  fxcopycolormapandfree
integer*4 display
integer*4 cmap
integer*4 colormap

colormap = fxcopycolormapandfree( display, cmap)
```

display Specifies the connection to the X Server.

cmap Specifies the colormap identification.

fxcopycolormapandfree obtains a new colormap when allocating out of a previously shared colormap which failed due to resource exhaustion. Resource exhaustion occurs when too many cells or planes are in use in the original colormap.

Return value *colormap* contains a colormap identification associated with the colormap created by the function for the screen on which the window resides.

fxcopycolormapandfree can generate the errors **BadAlloc** and **BadColor**.

For more information about the C language **Xlib** routine called by the **FXlib** binding routine **fxcopycolormapandfree**, see the **XCopyColormapAndFree** routine in "Xlib Functions."

¦ Copyright IBM Corp. 1985, 1991

3.5.47 fxcopygc

external fxcopygc
integer*4 *display*
integer*4 *src*
integer*4 *valuemaskcopy*
integer*4 *dest*

call fxcopygc(*display, src, valuemaskcopy, dest*)

display Specifies the connection to the X Server.

src Specifies the address of type **GC** that has the components of the source graphics context.

valuemaskcopy Specifies which components in the source graphics context are to be copied to the destination graphics context. This argument is a bitwise inclusive OR of valid **GC** component mask bits.

dest Specifies the address of type **GC** that has the components of the destination graphics context.

fxcopygc copies components from a source graphics context to a destination graphics context.

This routine can generate the errors **BadGC**, **BadAlloc**, and **BadValue**.

For more information about the C language **Xlib** routine called by the **FXlib** binding routine **fxcopygc**, see the **XCopyGC** routine in "Xlib Functions."

¦ Copyright IBM Corp. 1985, 1991

3.5.48 fxcopyplane

```
external  fxcopyplane
integer*4 display
integer*4 src, dest
integer*4 gc
integer*4 srcx, srcy, width, height
integer*4 destx, desty, plane

call fxcopyplane( display, src, dest, gc, srcx, srcy, width,
*                      height destx, desty, plane)
```

display Specifies the connection to the X Server.

src Specifies the source rectangles.

dest Specifies the destination rectangles.

gc Specifies an address of type **GC** that has the graphics context.

srcx, srcy Specify the *x* and *y* coordinates of the source rectangle relative to its origin. These coordinates specify the upper-left corner of the source rectangle.

width Specifies the width of both the source and destination rectangles.

height Specifies the height of both the source and destination rectangles.

destx, desty Specify the *x* and *y* coordinates of the destination rectangle relative to its origin. These coordinates specify the upper-left corner of the destination rectangle.

fxcopyplane copies a single bit-plane of the specified drawable. The drawable can have different depths.

This routine can generate the errors **BadDrawable**, **BadValue**, **BadGC**, and **BadMatch**.

For more information about the C language **Xlib** routine called by the **FXlib** binding routine **fxcopyplane**, see the **XCopyPlane** routine in "Xlib Functions."

¦ Copyright IBM Corp. 1985, 1991

3.5.49 fxcreatebitmapfromdata

```
integer*4 fxcreatebitmapfromdata
external  fxcreatebitmapfromdata
integer*4 display, drawable
integer*4 data
integer*4 width, height
integer*4 pixmap

pixmap = fxcreatebitmapfromdata( display, drawable, data, width, height)
```

display Specifies the connection to the X Server.

drawable Specifies the drawable.

data Specifies the location of the bitmap data.

width, height Specify the *width* and *height* dimensions of the bitmap to create.

fxcreatebitmapfromdata includes a bitmap written out by **XWriteBitmapFile** in a program directly, as opposed to reading it in every time at run time.

Return value *pixmap* is the id of the created bitmap.

For more information about the C language **Xlib** routine called by the **FXlib** binding routine **fxcreatebitmapfromdata**, see the **XCreateBitmapFromData** routine in "Xlib Functions."

¦ Copyright IBM Corp. 1985, 1991

3.5.50 fxcreatecolormap

integer*4 fxcreatecolormap
external fxcreatecolormap
integer*4 *display*
integer*4 *window*
integer*4 *visual*
integer*4 *alloc*
integer*4 *colormap*

colormap = **fxcreatecolormap**(*display, window, visual, alloc*)

display Specifies the connection to the X Server.

window Specifies the window ID to the window.

visual Specifies an address for type **Visual** supported on the screen.

alloc Specifies the colormap entries to be allocated.

fxcreatecolormap creates a colormap for the screen on which the window resides and returns the colormap ID. If *visual* is not supported by the screen, it returns a **BadMatch** error.

The variable *alloc* can be set to **AllocNone** or **AllocAll**.

Return value *colormap* contains a colormap ID associated with the colormap created by the function for the screen on which the window resides.

fxcreatecolormap can generate the event errors **BadWindow**, **BadAlloc**, **BadMatch**, and **BadValue**.

For more information about the C language **Xlib** routine called by the **FXlib** binding routine **fxcreatecolormap**, see the **XCreateColormap** routine in "Xlib Functions."

¦ Copyright IBM Corp. 1985, 1991

3.5.51 fxcreatefontcursor

```
integer*4 fxcreatefontcursor
external  fxcreatefontcursor
integer*4 display
integer*4 shape
integer*4 cursor

cursor = fxcreatefontcursor( display, shape)
```

display Specifies the connection to the X Server.

shape Specifies the shape for the standard cursor.

fxcreatefontcursor creates a cursor from a standard font.

Return value *cursor* is the cursor ID.

fxcreatefontcursor can generate the errors **BadAlloc**, **BadValue**, and **BadMatch**.

For more information about the C language **Xlib** routine called by the **FXlib** binding routine **fxcreatefontcursor**, see the **XCreateFontCursor** routine in "Xlib Functions."

¦ Copyright IBM Corp. 1985, 1991

3.5.52 fxcreategc

integer*4 fxcreategc
external fxcreategc
integer*4 *display*
integer*4 *drawable*
integer*4 *valuemaskcreate*
integer*4 *values*
integer*4 *gc*

gc = **fxcreategc**(*display, drawable, valuemaskcreate, values*)

display Specifies the connection to the X Server.

drawable Specifies the drawable.

valuemaskcreate Specifies which components in the **GC** are to be set using information in the type **XGCValues** data structure. This argument is a bitwise inclusive OR of valid **GC** component mask bits.

values Specifies the address of type **XGCValues** of the data structure.

fxcreategc creates a new **GC** for the specified drawable.

The variable *gc* contains the address of the created **GC** that can be used with any destination drawable having the same root and depth as the specified drawable.

fxcreategc can generate the error **BadMatch**.

For more information about the C language **Xlib** routine called by the **FXlib** binding routine **fxcreategc**, see the **XCreateGC** routine in "Xlib Functions."

¦ Copyright IBM Corp. 1985, 1991

3.5.53 fxcreateglyphcursor

```
integer*4 fxcreateglyphcursor
external  fxcreateglyphcursor
integer*4 display
integer*4 sourcefont, maskfont
integer*4 sourcecharacter, maskcharacter
integer*4 foregroundcolor, backgroundcolor
integer*4 cursor

cursor = fxcreateglyphcursor( display, sourcefont, maskfont,
*                                      sourcecharacter, maskcharacter,
*                                      foregroundcolor, backgroundcolor)
```

display Specifies the connection to the X Server.

sourcefont Specifies the font for the source glyph.

maskfont Specifies the font for the mask glyph. This variable can be set to **None**.

sourcecharacter Specifies the character glyph for the source.

maskcharacter Specifies the glyph character for the mask.

foregroundcolor Specifies the address of type **XColor** that has the RGB values for the foreground of the source.

backgroundcolor Specifies the address of type **XColor** that has the RGB values for the background of the source.

fxcreateglyphcursor creates a cursor from font glyphs.

This routine can generate the errors **BadAlloc**, **BadFont**, and **BadValue**.

For more information about the C language **Xlib** routine called by the **FXlib** binding routine **fxcreateglyphcursor**, see the **XCreateGlyphCursor** routine in "Xlib Functions."

¦ Copyright IBM Corp. 1985, 1991

3.5.54 fxcreateimage

```
integer*4 fxcreateimage
external  fxcreateimage
integer*4 display, visual, depth, format, offset
integer*4 data
integer*4 width, height, bitmappad, bytesperline
integer*4 image

image = fxcreateimage( display, visual, depth, format, offset,
*                               data, width, height, bitmappad, bytesperline)
```

display Specifies the connection to the X Server.

visual Specifies the address of the visual structure.

depth Specifies the *depth* of the image.

format Specifies the *format* for the image.

offset Specifies the number of pixels to ignore at the beginning of the scan line. This argument permits the rapid displaying of *image* without requiring each scan line to be shifted into position.

data Specifies the address of the image data.

width Specifies the *width* of the image in pixels.

height Specifies the *height* of the image in pixels.

bitmappad Specifies the quantum of a scan line. The start of one scan line is separated in client memory from the start of the next scan line by an integer multiple of the specified number of bits. This value must be a 8, 16, or 32.

bytesperline Specifies the number of bytes in the client image between the start of one scan line and the start of the next. If this value is a zero, **Xlib** assumes that the scan lines are contiguous in memory and calculates the value of *bytesperline*.

fxcreateimage allocates sufficient memory for an **XImage** structure.

The variable *format* can be **XYPixmap** or **ZPixmap**.

Return value *image* is the address of type **XImage** of the image structure created.

For more information about the C language **Xlib** routine called by the **FXlib** binding routine **fxcreateimage**, see the **XCreateImage** routine in "Xlib Functions."

¦ Copyright IBM Corp. 1985, 1991

3.5.55 fxcreatepixmap

integer*4 fxcreatepixmap
external fxcreatepixmap
integer*4 *display*
integer*4 *drawable*
integer*4 *width, height, depth*
integer*4 *pixmap*

pixmap = **fxcreatepixmap**(*display, drawable, width, height, depth*)

display Specifies the connection to the X Server.

drawable Specifies which screen the pixmap is created on.

width Specifies the width of the pixmap. Use non-zero values.

height Specifies the height of the pixmap. Use non-zero values.

depth Specifies the depth of the pixmap that must be supported by the root of the drawable specified.

fxcreatepixmap creates a pixmap of a specified size.

This routine can generate the errors **BadAlloc**, **BadDrawable**, and **BadValue**.

For more information about the C language **Xlib** routine called by the **FXlib** binding routine **fxcreatepixmap**, see the **XCreatePixmap** routine in "Xlib Functions."

¦ Copyright IBM Corp. 1985, 1991

3.5.56 fxcreatepixmapcursor

```
integer*4 fxcreatepixmapcursor
external  fxcreatepixmapcursor
integer*4 display
integer*4 source, mask
integer*4 foregroundcolor, backgroundcolor
integer*4 x, y
integer*4 cursor

cursor = fxcreatepixmapcursor( display, source, mask,
*                                       foregroundcolor, backgroundcolor, x
```

display	Specifies the connection to the X Server.
source	Specifies the shape of the source cursor.
mask	Specifies the source bits of the cursor that are to be displayed. This variable can be set to **None**.
foregroundcolor	Specifies the address of type **XColor** that has the RGB values for the foreground of the source.
backgroundcolor	Specifies the address of type **XColor** that has the RGB values for the background of the source.
x, y	Specify the *x* and *y* coordinates that indicate the hot spot relative to the origin of the source. The hot spot must be a point within the source.

fxcreatepixmapcursor creates a cursor from two bitmaps.

This routine can generate the errors **BadAlloc BadMatch**, and **BadPixmap**.

For more information about the C language **Xlib** routine called by the **FXlib** binding routine **fxcreatepixmapcursor**, see the **XCreatePixmapCursor** routine in "Xlib Functions."

¦ Copyright IBM Corp. 1985, 1991

3.5.57 fxcreatepixmapfrombitmapdata

```
integer*4 fxcreatepixmapfrombitmapdata
external  fxcreatepixmapfrombitmapdata
integer*4 display, drawable
integer*4 data
integer*4 width, height
integer*4fg, bg, depth
integer*4 pixmap

pixmap = fxcreatepixmapfrombitmapdata( display, drawable, data, width, height,
*                                                              fg, bg, depth)
```

display Specifies the connection to the X Server.

drawable Specifies the drawable.

data Specifies the data in bitmap format.

width, height Specify the *width* and *height* of the pixmap to create.

fg, bg Specify the *foreground* and *background* pixel values to use.

depth Specifies the *depth* of the pixmap.

fxcreatepixmapfrombitmapdata creates a pixmap of the given depth. Then, it does a bitmap-format **XPutImage** of the data into it.

Return value *pixmap* is the id of the created pixmap.

For more information about the C language **Xlib** routine called by the **FXlib** binding routine **fxcreatepixmapfrombitmapdata**, see the **XCreateBitmapFromBitmapData** routine in "Xlib Functions."

¦ Copyright IBM Corp. 1985, 1991

3.5.58 fxcreateregion

```
integer*4 fxcreateregion
external  fxcreateregion
integer*4 newregion

newregion = fxcreateregion()
```

fxcreateregion creates a new empty region.

Return value *newregion* is an address of type **_XRegion** of the created region structure.

For more information about the C language **Xlib** routine called by the **FXlib** binding routine **fxcreateregion**, see the **XCreateRegion** routine in "Xlib Functions."

¦ Copyright IBM Corp. 1985, 1991

3.5.59 fxcreatesimplewindow

```
integer*4 fxcreatesimplewindow
external  fxcreatesimplewindow
integer*4 display
integer*4 parent
integer*4 x, y
integer*4 width, height, bdrwidth
integer*4 border, background
integer*4 id

id = fxcreatesimplewindow( display, parent, x, y, width, height,
*                                    bdrwidth, border, background)
```

display Specifies the connection to the X Server.

parent Specifies the *parent* window ID.

x, y Specify the *x* and *y* coordinates. These coordinates are the top-left, outside corner of the border of the subwindow. The *x* and *y* coordinates are relative to the inside of the borders of the parent window.

width, height Specify the *width* and *height* dimensions of the subwindow. These dimensions do not include the border of the subwindow that are outside of the window. The dimensions for *width* and *height* must be nonzero.

bdrwidth Specifies the border width of the subwindow in pixels.

background Specifies the pixel value set for the background of the subwindow.

fxcreatesimplewindow creates an unmapped **InputOutput** subwindow of the specified parent window.

The variable *id* contains the window ID of the subwindow.

fxcreatesimplewindow can generate the error **BadMatch**.

For more information about the C language **Xlib** routine called by the **FXlib** binding routine **fxcreatesimplewindow**, see the **XCreateSimpleWindow** routine in "Xlib Functions."

¦ Copyright IBM Corp. 1985, 1991

3.5.60 fxcreatewindow

```
integer*4 fxcreatewindow
external  fxcreatewindow
integer*4 display
integer*4 parent
integer*4 x, y
integer*4 width, height, bdrwidth, depth
integer*4 class, visual, valuemask, attribs
integer*4 id

id = fxcreatewindow( display, parent, x, y, width, height, bdrwidth,
*                            depth, class, visual, valuemask, attribs)
```

display Specifies the connection to the X Server.

parent Specifies the parent window ID.

x, y Specify the *x* and *y* coordinates. These coordinates are the top-left, outside corner of the borders of the subwindow. These coordinates are relative to the inside of the borders of the parent window.

width, height Specify the *width* and *height*. These dimensions are the inside dimensions of the subwindow. These dimensions do not include the borders of the subwindows that are outside of the window. These dimensions must be nonzero. Otherwise, a **BadMatch** error is returned.

bdrwidth Specifies the border width of the subwindow in pixels.

depth Depth is zero if *class* is **InputOutput**, or, the depth is taken from the parent if *class* is **CopyFromParent**.

class Specifies the class of the subwindow.

visual Specifies the visual type.

valuemask Specifies which window attributes are defined in the attributes argument. This mask is a bitwise inclusive OR of valid attributes mask bits.

attributes Address of the window attributes that are set when the subwindow is created.

fxcreatewindow creates an unmapped subwindow for a specified parent window. It returns the window ID of the subwindow and causes the X Server to generate a **CreateNotify** event.

The variable *id* contains the window ID of the subwindow.

For more information about the C language **Xlib** routine called by the **FXlib** binding routine **fxcreatewindow**, see the **XCreateWindow** routine in "Xlib Functions."

¦ Copyright IBM Corp. 1985, 1991

3.5.61 fxdefaultcolormap

```
integer*4  fxdefaultcolormap
external   fxdefaultcolormap
integer*4  display
integer*4  screen
integer*4  rc

rc = fxdefaultcolormap( display, screen)
```

display Specifies the connection to the X Server.

screen Specifies the screen.

fxdefaultcolormap obtains the default colormap identification for allocation on the screen specified.

Return value *rc* is the default colormap ID for allocation on the specified screen. Most routine allocations of color should be made out of this colormap.

For more information about the C language **Xlib** routine called by the **FXlib** binding routine **fxdefaultcolormap**, see the **XDefaultColormap** routine in "Xlib Functions."

¦ Copyright IBM Corp. 1985, 1991

3.5.62 fxdefaultcolormapofscreen

```
integer*4  fxdefaultcolormapofscreen
external   fxdefaultcolormapofscreen
integer*4  screen
integer*4  rc

rc = fxdefaultcolormapofscreen( screen)
```

screen Specifies the address of type **Screen** of the screen of the display.

fxdefaultcolormapofscreen returns the default colormap of the screen specified.

Return value *rc* is the default colormap of the specified screen.

For more information about the C language **Xlib** routine called by the **FXlib** binding routine **fxdefaultcolormapofscreen**, see the **XDefaultColormapOfScreen** routine in "Xlib Functions."

¦ Copyright IBM Corp. 1985, 1991

3.5.63 fxdefaultdepth

```
integer*4  fxdefaultdepth
external   fxdefaultdepth
integer*4  display
integer*4  screen
integer*4  rc

rc = fxdefaultdepth( display, screen)
```

display Specifies the connection to the X Server.

screen Specifies the screen.

fxdefaultdepth obtains the default depth of the default root window for the screen specified. Other depths may also be supported on this screen.

Return value *rc* is the depth (number of planes) of the default root window for the specified screen.

For more information about the C language **Xlib** routine called by the **FXlib** binding routine **fxdefaultdepth**, see the **XDefaultDepth** routine in "Xlib Functions."

¦ Copyright IBM Corp. 1985, 1991

3.5.64 fxdefaultdepthofscreen

```
integer*4  fxdefaultdepthofscreen
external   fxdefaultdepthofscreen
integer*4  screen
integer*4  rc

rc = fxdefaultdepthofscreen( screen)
```

screen Specifies the address of type **Screen** of the screen of the display.

fxdefaultdepthofscreen returns the default depth of the screen specified.

Return value *rc* is the default depth (number of planes) of the screen specified.

For more information about the C language **Xlib** routine called by the **FXlib** binding routine **fxdefaultdepthofscreen**, see the **XDefaultDepthOfScreen** routine in "Xlib Functions."

¦ Copyright IBM Corp. 1985, 1991

3.5.65 fxdefaultgc

```
integer*4 fxdefaultgc
external  fxdefaultgc
integer*4  display
integer*4  screen
integer*4  defgc

defgc = fxdefaultgc( display, screen)
```

display Specifies the connection to the X Server.

screen Specifies the screen.

fxdefaultgc obtains the default graphic context (**GC**) of the default root window for the screen specified.

Return value *defgc* is an address of type **GC** that contains the default graphics context structure for the display and screen specified.

For more information about the C language **Xlib** routine called by the **FXlib** binding routine **fxdefaultgc**, see the **XDefaultGC** routine in "Xlib Functions."

¦ Copyright IBM Corp. 1985, 1991

3.5.66 fxdefaultgcofscreen

```
integer*4  fxdefaultgcofscreen
external   fxdefaultgcofscreen
integer*4  screen
integer*4  rc

rc = fxdefaultgcofscreen( screen)
```

screen Specifies the address of type **Screen** of the screen of the display.

fxdefaultgcofscreen returns the default graphic context (**GC**) of the screen specified.

Return value *rc* is the default graphic context of the screen specified.

For more information about the C language **Xlib** routine called by the **FXlib** binding routine **fxdefaultgcofscreen**, see the **XDefaultGCOfScreen** routine in "Xlib Functions."

¦ Copyright IBM Corp. 1985, 1991

3.5.67 fxdefaultrootwindow

```
integer*4  fxdefaultrootwindow
external   fxdefaultrootwindow
integer*4  display
integer*4  rc

rc = fxdefaultrootwindow( display)
```

display Specifies the connection to the X Server.

fxdefaultrootwindow obtains the root window for the default screen.

Return value *rc* is the value of the root window for the default screen.

For more information about the C language **Xlib** routine called by the **FXlib** binding routine **fxdefaultrootwindow**, see the **XDefaultRootWindow** routine in "Xlib Functions."

¦ Copyright IBM Corp. 1985, 1991

3.5.68 fxdefaultscreen

```
integer*4  fxdefaultscreen
external   fxdefaultscreen
integer*4  display
integer*4  defscrdisplay

defscrdisplay =  fxdefaultscreen( display)
```

display Specifies the connection to the X Server.

fxdefaultscreen obtains the default screen referenced in the **fxopendisplay** routine.

Return value *defscrdisplay* is an address of type **Screen** of the default screen information structure for the display specified.

For more information about the C language **Xlib** routine called by the **FXlib** binding routine **fxdefaultscreen**, see the **XDefaultScreen** routine in "Xlib Functions."

¦ Copyright IBM Corp. 1985, 1991

3.5.69 fxdefaultscreenofdisplay

```
integer*4  fxdefaultscreenofdisplay
external   fxdefaultscreenofdisplay
integer*4  display
integer*4  rc

rc = fxdefaultscreenofdisplay( display)
```

display Specifies the connection to the X Server.

fxdefaultscreenofdisplay returns the default screen of the display specified.

Return value *rc* is the default screen of the display specified.

For more information about the C language **Xlib** routine called by the **FXlib** binding routine **fxdefaultscreenofdisplay**, see the **XDefaultScreenOfDisplay** routine in "Xlib Functions."

¦ Copyright IBM Corp. 1985, 1991

3.5.70 fxdefaultvisual

```
integer*4  fxdefaultvisual
external   fxdefaultvisual
integer*4  display
integer*4  screen
integer*4  defvisual

defvisual = fxdefaultvisual( display, screen)
```

display Specifies the connection to the X Server.

screen Specifies the screen.

fxdefaultvisual obtains the default visual type for the specified screen.

Return value *defvisual* is an address of type **Visual** that contains the default visual type for the display and screen specified.

For more information about the C language **Xlib** routine called by the **FXlib** binding routine **fxdefaultvisual**, see the **XDefaultVisual** routine in "Xlib Functions."

¦ Copyright IBM Corp. 1985, 1991

3.5.71 fxdefaultvisualofscreen

```
integer*4  fxdefaultvisualofscreen
external   fxdefaultvisualofscreen
integer*4  screen
integer*4  rc

rc = fxdefaultvisualofscreen( screen)
```

screen Specifies the address of type **Screen** of the screen of the display.

fxdefaultvisualofscreen returns the default visual of the screen specified.

Return value *rc* is the default visual of the screen specified.

For more information about the C language **Xlib** routine called by the **FXlib** binding routine **fxdefaultscreenofdisplay**, see the **XDefaultScreenOfDisplay** routine in "Xlib Functions."

¦ Copyright IBM Corp. 1985, 1991

3.5.72 fxdefinecursor

external fxdefinecursor
integer*4 *display*
integer*4 *window, cursor*

call fxdefinecursor(*display, window, cursor*)

display Specifies the connection to the X Server.

window Specifies the window ID.

cursor Specifies the cursor to be displayed when the pointer is in the specified window. This variable can be set to **None** if no cursor is to be displayed.

fxdefinecursor defines which cursor will be used in a window.

This routine can generate the errors **BadWindow**, **BadAlloc**, and **BadCursor**.

For more information about the C language **Xlib** routine called by the **FXlib** binding routine **fxdefinecursor**, see the **XDefineCursor** routine in "Xlib Functions."

¦ Copyright IBM Corp. 1985, 1991

3.5.73 fxdeletecontext

integer*4 fxdeletecontext
external fxdeletecontext
integer*4 *display, window, context*
integer*4 *status*

status = **fxdeletecontext**(*display, window, context*)

display Specifies the connection to the X Server.

window Specifies the window associated with the data.

context Specifies the context type to which the data belongs.

fxdeletecontext deletes an entry for a given window and type.

For more information about the C language **Xlib** routine called by the **FXlib** binding routine **fxdeletecontext**, see the **XDeleteContext** routine in "Xlib Functions."

¦ Copyright IBM Corp. 1985, 1991

3.5.74 fxdeletemodifiermapentry

```
integer*4 fxdeletemodifiermapentry
external fxdeletemodifiermapentry
integer*4 modmap
integer*4 keysyment
integer*4 modifier
integer*4 keymap

keymap = fxdeletemodifiermapentry( modmap, keysyment, modifier)
```

modmap Specifies the address of type **XModifierKeymap** of the keymap modifier data structure.

keysyment Specifies the keysyms.

modifier Specifies the modifier.

fxdeletemodifiermapentry deletes an entry from an **XModifierKeymap** structure.

For more information about the C language **Xlib** routine called by the **FXlib** binding routine **fxdeletemodifiermapentry**, see the **XDeleteModifiermapEntry** routine in "Xlib Functions."

¦ Copyright IBM Corp. 1985, 1991

3.5.75 fxdeleteproperty

external fxdeleteproperty
integer*4 *display*
integer*4 *window*
integer*4 *property*

call fxdeleteproperty(*display, window, property*)

display Specifies the connection to the X Server.

window Specifies the window ID for the window whose property is to be deleted.

property Specifies the property atom.

fxdeleteproperty deletes a property for the specified window.

The property specified is deleted if it was defined on the specified window. **fxdeleteproperty** causes the X Server to generate a **PropertyNotify** event on the window unless the property does not exist.

This routine can generate the errors **BadAtom** and **BadWindow**.

For more information about the C language **Xlib** routine called by the **FXlib** binding routine **fxdeleteproperty**, see the **XDeleteProperty** routine in "Xlib Functions."

¦ Copyright IBM Corp. 1985, 1991

3.5.76 fxdestroyimage

external fxdestroyimage
integer*4 *ximage*

call fxdestroyimage(*ximage*)

ximage Specifies the address of type **XImage** of the image structure.

fxdestroyimage deallocates the memory allocated in a previous call to **fxcreateimage**.

For more information about the C language **Xlib** routine called by the **FXlib** binding routine **fxdestroyimage**, see the **XDestroyImage** routine in "Xlib Functions."

¦ Copyright IBM Corp. 1985, 1991

3.5.77 fxdestroyregion

external fxdestroyregion
integer*4 *r*

call fxdestroyregion(*r*)

r Specifies the address of type **_XRegion** of the region structure.

fxdestroyregion deallocates the storage associated with a specified region.

For more information about the C language **Xlib** routine called by the **FXlib** binding routine **fxdestroyregion**, see the **XDestroyRegion** routine in "Xlib Functions."

¦ Copyright IBM Corp. 1985, 1991

3.5.78 fxdestroysubwindows

```
external  fxdestroysubwindows
integer*4 display
integer*4 window

call fxdestroysubwindows( display, window)
```

display Specifies the connection to the X Server.

window Specifies the window ID.

fxdestroysubwindows destroys all subwindows of a specified window.

For more information about the C language **Xlib** routine called by the **FXlib** binding routine **fxdestroysubwindows**, see the **XDestroySubwindows** routine in "Xlib Functions."

¦ Copyright IBM Corp. 1985, 1991

3.5.79 fxdestroywindow

```
external  fxdestroywindow
integer*4 display
integer*4 window

call fxdestroywindow( display, window)
```

display Specifies the connection to the X Server.

window Specifies the window ID.

fxdestroywindow destroys a window and all of its subwindows.

For more information about the C language **Xlib** routine called by the **FXlib** binding routine **fxdestroywindow**, see the **XDestroyWindow** routine in "Xlib Functions."

¦ Copyright IBM Corp. 1985, 1991

3.5.80 fxdisableaccesscontrol

external fxdisableaccesscontrol
integer*4 *display*

call fxdisableaccesscontrol(*display*)

display Specifies the connection to the X Server.

fxdisableaccesscontrol disables access control.

This routine can generate the error **BadAccess**.

For more information about the C language **Xlib** routine called by the **FXlib** binding routine **fxdisableaccesscontrol**, see the **XDisableAccessControl** routine in "Xlib Functions."

¦ Copyright IBM Corp. 1985, 1991

3.5.81 fxdisplaycells

```
integer*4  fxdisplaycells
external   fxdisplaycells
integer*4  display
integer*4  screen
integer*4  rc

rc = fxdisplaycells( display, screen)
```

display Specifies the connection to the X Server.

screen Specifies the screen.

fxdisplaycells obtains the number of entries in the default colormap.

Return value *rc* is the number of entries in the default colormap.

For more information about the C language **Xlib** routine called by the **FXlib** binding routine **fxdisplaycells**, see the **XDisplayCells** routine in "Xlib Functions."

¦ Copyright IBM Corp. 1985, 1991

3.5.82 fxdisplayheight

```
integer*4  fxdisplayheight
external   fxdisplayheight
integer*4  display
integer*4  screen
integer*4  rc

rc = fxdisplayheight( display, screen)
```

display Specifies the connection to the X Server.

screen Specifies the screen.

fxdisplayheight returns an integer that describes the height of the screen in pixels.

Return value *rc* is the height of the screen in pixels.

For more information about the C language **Xlib** routine called by the **FXlib** binding routine **fxdisplayheight**, see the **XDisplayHeight** routine in "Xlib Functions."

¦ Copyright IBM Corp. 1985, 1991

3.5.83 fxdisplayheightmm

```
integer*4  fxdisplayheightmm
external   fxdisplayheightmm
integer*4  display
integer*4  screen
integer*4  rc

rc = fxdisplayheightmm( display, screen)
```

display Specifies the connection to the X Server.

screen Specifies the screen.

fxdisplayheightmm returns an integer that describes the height of the screen in millimeters.

Return value *rc* is the height of the screen in millimeters.

For more information about the C language **Xlib** routine called by the **FXlib** binding routine **fxdisplayheightmm**, see the **XDisplayHeightMM** routine in "Xlib Functions."

¦ Copyright IBM Corp. 1985, 1991

3.5.84 fxdisplaykeycodes

external fxdisplaykeycodes
integer*4 *display*
integer*4 *min_keycodes_return*
integer*4 *max_keycodes_return*

call fxdisplaykeycodes(*display,min_keycodes_return,max_keycodes_return*)

display Specifies the connection to the X Server.

min_keycodes_return Returns the minimum number of KeyCodes.

max_keycodes_return Returns the maximum number of KeyCodes.

fxdisplaykeycodes returns the minimum and the maximum number of KeyCodes supported by the specified display. The *min_keycodes_return* is is never less than 8. The *max_keycodes_return* is never more than 255. Not all KeyCodes in this range must have corresponding keys.

For more information about the C language **Xlib** routine called by the **FXlib** binding routine **fxdisplaykeycodes**, see the **XDisplayKeycodes** routine in "Xlib Functions."

¦ Copyright IBM Corp. 1985, 1991

3.5.85 fxdisplaymotionbuffersize

external fxdisplaymotionbuffersize
integer*4 *display*
integer*4 *size*

size = **fxdisplaymotionbuffersize**(*display*)

display Specifies the connection to the X Server.

fxdisplaymotionbuffersize returns the size of the motion buffer. It retains the recent history of the pointer motion. It is done so to a finer granularity than is reported by **MotionNotify** events. **XGetMotionEvents** makes this history available.

For more information about the C language **Xlib** routine called by the **FXlib** binding routine **fxdisplaymotionbuffersize**, see the **XDisplayMotionBufferSize** routine in "Xlib Functions."

¦ Copyright IBM Corp. 1985, 1991

3.5.86 fxdisplayname

character*256 fxdisplayname
external fxdisplayname
character*256 *rc*
character*256 *string*

rc = **fxdisplayname**(*string*)

string Specifies the character string.

fxdisplayname reports an error to the user when the requested display does not exist.

Return value *rc* is the name of the display.

For more information about the C language **Xlib** routine called by the **FXlib** binding routine **fxdisplayname**, see the **XDisplayName** routine in "Xlib Functions."

¦ Copyright IBM Corp. 1985, 1991

3.5.87 fxdisplayofscreen

```
integer*4  fxdisplayofscreen
external   fxdisplayofscreen
integer*4  display
integer*4  dispscreen

dispscreen = fxdisplayofscreen( screen)
```

screen Specifies the address of type **Screen** of the screen of the display.

fxdisplayofscreen returns the display of the screen specified.

Return value *dispscreen* is an address of type **Display** of the display structure for the *screen* specified.

For more information about the C language **Xlib** routine called by the **FXlib** binding routine **fxdisplayofscreen**, see the **XDisplayOfScreen** routine in "Xlib Functions."

¦ Copyright IBM Corp. 1985, 1991

3.5.88 fxdisplayplanes

```
integer*4  fxdisplayplanes
external   fxdisplayplanes
integer*4  display
integer*4  screen
integer*4  rc

rc = fxdisplayplanes( display, screen)
```

display Specifies the connection to the X Server.

screen Specifies the screen.

fxdisplayplanes obtains the depth of the root window of the screen specified.

Return value *rc* is the depth (number of planes) of the root window of the specified screen.

For more information about the C language **Xlib** routine called by the **FXlib** binding routine **fxdisplayplanes**, see the **XDisplayPlanes** routine in "Xlib Functions."

¦ Copyright IBM Corp. 1985, 1991

3.5.89 fxdisplaystring

```
character*256   fxdisplaystring
external  fxdisplaystring
integer*4       display
character*256   displaystring

displaystring = fxdisplaystring( display)
```

display Specifies the connection to the X Server.

fxdisplaystring obtains the string that was passed to **fxopendisplay** when the current display was opened.

Return value *displaystring* is the name of the current display (the string passed to **fxopendisplay**).

For more information about the C language **Xlib** routine called by the **FXlib** binding routine **fxdisplaystring**, see the **XDisplayString** routine in "Xlib Functions."

¦ Copyright IBM Corp. 1985, 1991

3.5.90 fxdisplaywidth

```
integer*4  fxdisplaywidth
external   fxdisplaywidth
integer*4  display
integer*4  screen
integer*4  rc
rc = fxdisplaywidth( display, screen)
```

display Specifies the connection to the X Server.

screen Specifies the screen.

fxdisplaywidth returns an integer that describes the width of the screen in pixels.

Return value *rc* is the width of the screen in pixels.

For more information about the C language **Xlib** routine called by the **FXlib** binding routine **fxdisplaywidth**, see the **XDisplayWidth** routine in "Xlib Functions."

¦ Copyright IBM Corp. 1985, 1991

3.5.91 fxdisplaywidthmm

```
integer*4  fxdisplaywidthmm
external   fxdisplaywidthmm
integer*4  display
integer*4  screen
integer*4  rc

rc = fxdisplaywidthmm( display, screen)
```

display Specifies the connection to the X Server.

screen Specifies the screen.

fxdisplaywidthmm returns an integer that describes the width of the screen in millimeters.

Return value *rc* is the width of the screen in millimeters.

For more information about the C language **Xlib** routine called by the **FXlib** binding routine **fxdisplaywidthmm**, see the **XDisplayWidthMM** routine in "Xlib Functions."

¦ Copyright IBM Corp. 1985, 1991

3.5.92 fxdoesbackingstore

```
integer*4  fxdoesbackingstore
external   fxdoesbackingstore
integer*4  screen
integer*4  rc

rc = fxdoesbackingstore( screen)
```

screen Specifies the address of type **Screen** of the screen of the display.

fxdoesbackingstore returns a value indicating whether the screen supports backing stores.

Return value *rc* indicates whether the screen supports backing stores. The value can be **whenmapped**, **notuseful**, or **always**.

For more information about the C language **Xlib** routine called by the **FXlib** binding routine **fxdoesbackingstore**, see the **XDoesBackingStore** routine in "Xlib Functions."

¦ Copyright IBM Corp. 1985, 1991

3.5.93 fxdoessaveunders

```
integer*4  fxdoessaveunders
external   fxdoessaveunders
integer*4  screen
integer*4  rc

rc = fxdoessaveunders( screen)
```

screen Specifies the address of type **Screen** of the screen of the display.

fxdoessaveunders returns a value indicating whether the screen supports saveunders.

Return value *rc* is TRUE if the screen supports saveunders, or FALSE if it does not.

For more information about the C language **Xlib** routine called by the **FXlib** binding routine **fxdoessaveunders**, see the **XDoesSaveUnders** routine in "Xlib Functions."

¦ Copyright IBM Corp. 1985, 1991

3.5.94 fxdrawarc

external fxdrawarc
integer*4 *display*
integer*4 *drawable, gc*
integer*4 *x, y, width, height, angle1, angle2*

call fxdrawarc(*display, drawable, gc, x, y, width, height, angle1,*
** angle2*)

display Specifies the connection to the X Server.

drawable Specifies the drawable.

gc Specifies an address of type **GC** that has the graphics context.

x, y Specify the *x* and *y* coordinates of the arc. These coordinates are relative to the origin of the drawable. These coordinates specify the upper-left corner of the rectangle.

width, height Specify the *width* and *height* that are the major and minor axes of the arc.

angle1 Specifies the start of the arc relative to the 3 o'clock position from the center in units of degrees multiplied by 64.

angle2 Specifies the path and extent of the arc relative to the start of the arc in units of degrees multiplied by 64.

fxdrawarc draws a single arc.

This routine can generate the errors **BadDrawable**, **BadMatch**, and **BadGC**.

For more information about the C language **Xlib** routine called by the **FXlib** binding routine **fxdrawarc**, see the **XDrawArc** routine in "Xlib Functions."

¦ Copyright IBM Corp. 1985, 1991

3.5.95 fxdrawarcs

external fxdrawarcs
integer*4 *display*
integer*4 *drawable, gc*
integer*4 *arcs, narcs*

call fxdrawarcs(*display, drawable, gc, arcs, narcs*)

display Specifies the connection to the X Server.

drawable Specifies the drawable.

gc Specifies an address of type **GC** that has the graphics context.

arcs Specifies an address of type **XArc** of an array of arcs. The client must allocate the arc list and must free the list when it is no longer needed.

narcs Specifies the number of arcs in the array.

fxdrawarcs draws the multiple arcs in a specified drawable.

This routine can generate the errors **BadDrawable**, **BadMatch**, and **BadGC**.

For more information about the C language **Xlib** routine called by the **FXlib** binding routine **fxdrawarcs**, see the **XDrawArcs** routine in "Xlib Functions."

¦ Copyright IBM Corp. 1985, 1991

3.5.96 fxdrawimagestring

external fxdrawimagestring
integer*4 *display*
integer*4 *drawable, gc*
integer*4 *x, y*
character*256 *string*
integer*4 *length*

call fxdrawimagestring(*display, drawable, gc, x, y, string, length*)

display Specifies the connection to the X Server.

drawable Specifies the drawable.

gc Specifies an address of type **GC** that has the graphics context.

x, y Specify the *x* and *y* coordinates that define the baseline starting position for the origin of the initial character. These coordinates are relative to the origin of the specified drawable.

string Specifies the address of type **XChar2b** that has the character string of 8-bit characters.

length Specifies the number of characters in the string argument.

fxdrawimagestring draws 8-bit text characters when both foreground and background bits of each character are painted.

This routine can generate the errors **BadDrawable**, **BadMatch**, and **BadGC**.

For more information about the C language **Xlib** routine called by the **FXlib** binding routine **fxdrawimagestring**, see the **XDrawImageString** routine in "Xlib Functions."

¦ Copyright IBM Corp. 1985, 1991

3.5.97 fxdrawimagestring16

external fxdrawimagestring16
integer*4 *display*
integer*4 *drawable, gc*
integer*4 *x, y, string, length*

call fxdrawimagestring16(*display, drawable, gc, x, y, string, length*)

display Specifies the connection to the X Server.

drawable Specifies the drawable.

gc Specifies an address of type **GC** that has the graphics context.

x, y Specify the *x* and *y* coordinates that define the baseline starting position for the origin of the initial character. These coordinates are relative to the origin of the specified drawable.

string Specifies the character string of 16-bit characters.

length Specifies the number of characters in the string argument.

fxdrawimagestring16 draws 16-bit text characters when both foreground and background bits of each character are painted.

This routine can generate the errors **BadDrawable**, **BadMatch**, and **BadGC**.

For more information about the C language **Xlib** routine called by the **FXlib** binding routine **fxdrawimagestring16**, see the **XDrawImageString16** routine in "Xlib Functions."

¦ Copyright IBM Corp. 1985, 1991

3.5.98 fxdrawline

external fxdrawline
integer*4 *display*
integer*4 *drawable, gc*
integer*4 *x1, y1, x2, y2*

call fxdrawline(*display, drawable, gc, x1, y1, x2, y2*)

display Specifies the connection to the X Server.

drawable Specifies the drawable.

gc Specifies an address of type **GC** that has the graphics context.

x1, y1, x2, y2 Specify the points used to connect the line.

fxdrawline draws a single line between two points in the specified drawable. It draws a line connecting point *x1, y1* to point *x2, y2*.

This routine can generate errors **BadDrawable**, **BadMatch**, and **BadGC**.

For more information about the C language **Xlib** routine called by the **FXlib** binding routine **fxdrawline**, see the **XDrawLine** routine in "Xlib Functions."

¦ Copyright IBM Corp. 1985, 1991

3.5.99 fxdrawlines

external fxdrawlines
integer*4 *display*
integer*4 *drawable, gc*
integer*4 *points, npoints, mode*

call fxdrawlines(*display, drawable, gc, points, npoints, mode*)

display Specifies the connection to the X Server.

drawable Specifies the drawable.

gc Specifies an address of type **GC** that has the graphics context.

points Specifies an address of type **XPoint** of an array of points. The client must allocate the line list and must free the line list when it is no longer needed.

npoints Specifies the number of points in the array.

mode Specifies the coordinate mode.

fxdrawlines draws multiple lines in the specified drawable.

Possible values for *mode* are **CoordModeOrigin**, which treats a coordinate as related to the origin, and **CoordModePrevious**, which treats all coordinates after the first as relative to the previous point.

This routine can generate errors **BadDrawable**, **BadMatch**, **BadValue**, and **BadGC**.

For more information about the C language **Xlib** routine called by the **FXlib** binding routine **fxdrawlines**, see the **XDrawLines** routine in "Xlib Functions."

¦ Copyright IBM Corp. 1985, 1991

3.5.100 fxdrawpoint

external fxdrawpoint
integer*4 *display*
integer*4 *drawable*
integer*4 *gc*
integer*4 *x, y*

call fxdrawpoint(*display, drawable, gc, x, y*)

display Specifies the connection to the X Server.

drawable Specifies the drawable.

gc Specifies an address of type **GC** that has the graphics context.

x, y Specify the *x* and *y* coordinates where the point is drawn.

fxdrawpoint draws a single point.

This routine can generate errors **BadDrawable**, **BadMatch**, and **BadGC**.

For more information about the C language **Xlib** routine called by the **FXlib** binding routine **fxdrawpoint**, see the **XDrawPoint** routine in "Xlib Functions."

¦ Copyright IBM Corp. 1985, 1991

3.5.101 fxdrawpoints

external fxdrawpoints
integer*4 *display*
integer*4 *drawable, gc, points*
integer*4 *npoints, mode*

call fxdrawpoints(*display, drawable, gc, points, npoints, mode*)

display Specifies the connection to the X Server.

drawable Specifies the drawable.

gc Specifies an address of type **GC** that has the graphics context.

points Specifies an address of type **XPoint** of an array of points. The client must allocate the point list and must free the point list when it is no longer needed.

npoints Specifies the number of points in the array.

mode Specifies the coordinate mode.

fxdrawpoints draws multiple points in the specified drawable.

Possible values for the variable *coordinate mode* are **CoordModeOrigin**, which treats a coordinate as related to the origin, and **CoordModePrevious**, which treats all coordinates after the first as relative to the previous point.

This routine can generate errors **BadDrawable**, **BadMatch**, **BadValue**, and **BadGC**.

For more information about the C language **Xlib** routine called by the **FXlib** binding routine **fxdrawpoints**, see the **XDrawPoints** routine in "Xlib Functions."

¦ Copyright IBM Corp. 1985, 1991

3.5.102 fxdrawrectangle

external fxdrawrectangle
integer*4 *display*
integer*4 *drawable, gc*
integer*4 *x, y, width, height*

call fxdrawrectangle(*display, drawable, gc, x, y, width, height*)

display Specifies the connection to the X Server.

drawable Specifies the drawable.

gc Specifies an address of type **GC** that has the graphics context.

x, y Specify the *x* and *y* coordinates that define the upper-left corner of the rectangle.

width, height Specify *width* and *height* that defines the outline of the rectangle.

fxdrawrectangle draws the outline of a single rectangle.

This function can generate errors **BadDrawable**, **BadMatch**, and **BadGC**.

For more information about the C language **Xlib** routine called by the **FXlib** binding routine **fxdrawrectangle**, see the **XDrawRectangle** routine in "Xlib Functions."

¦ Copyright IBM Corp. 1985, 1991

3.5.103 fxdrawrectangles

external fxdrawrectangles
integer*4 *display*
integer*4 *drawable, gc*
integer*4 *rectangles, nrectangles*

call fxdrawrectangles(*display, drawable, gc, rectangles, nrectangles*)

display Specifies the connection to the X Server.

drawable Specifies the drawable.

gc Specifies an address of type **GC** that has the graphics context.

rectangles Specifies an address of **XRectangle** of an array of rectangles. The client must allocate the rectangle list and must free the rectangle list when it is no longer needed.

nrectangles Specifies the number of rectangles in the array.

fxdrawrectangles draws the outline of multiple rectangles in a specified drawable.

This routine can generate errors **BadDrawable**, **BadMatch**, and **BadGC**.

For more information about the C language **Xlib** routine called by the **FXlib** binding routine **fxdrawrectangles**, see the **XDrawRectangles** routine in "Xlib Functions."

¦ Copyright IBM Corp. 1985, 1991

3.5.104 fxdrawsegments

external fxdrawsegments
integer*4 *display*
integer*4 *drawable, gc*
integer*4 *segments, nsegments*

call fxdrawsegments(*display, drawable, gc, segments, nsegments*)

display Specifies the connection to the X Server.

drawable Specifies the drawable.

segments Specifies the address of type **XSegment** of an array of segments. The client must allocate the segment list and must free the segment list when it is no longer needed.

nsegments Specifies the number of segments in the array.

fxdrawsegments draws multiple, but not necessarily connected, lines in the specified drawable.

This routine can generate errors **BadDrawable**, **BadMatch**, and **BadGC**.

For more information about the C language **Xlib** routine called by the **FXlib** binding routine **fxdrawsegments**, see the **XDrawSegments** routine in "Xlib Functions."

¦ Copyright IBM Corp. 1985, 1991

3.5.105 fxdrawstring

external fxdrawstring
integer*4 *display*
integer*4 *drawable, gc*
integer*4 *x, y*
character*256 *string*
integer*4 *length*

call fxdrawstring(*display, drawable, gc, x, y, string, length*)

display Specifies the connection to the X Server.

drawable Specifies the drawable.

gc Specifies an address of type **GC** that has the graphics context.

x, y Specify the *x* and *y* coordinates that define the baseline starting position for the origin of the initial character. These coordinates are relative to the origin of the specified drawable.

string Specifies the character string of 8-bit characters.

length Specifies the number of characters in the string argument.

fxdrawstring draws 8-bit text characters.

This routine can generate the errors **BadDrawable**, **BadMatch**, **BadFont**, and **BadGC**.

For more information about the C language **Xlib** routine called by the **FXlib** binding routine **fxdrawstring**, see the **XDrawString** routine in "Xlib Functions."

¦ Copyright IBM Corp. 1985, 1991

3.5.106 fxdrawstring16

external fxdrawstring16
integer*4 *display*
integer*4 *drawable, gc*
integer*4 *x, y, string, length*

call fxdrawstring16(*display, drawable, gc, x, y, string, length*)

display Specifies the connection to the X Server.

drawable Specifies the drawable.

gc Specifies an address of type **GC** that has the graphics context.

x, y Specify the *x* and *y* coordinates that define the baseline starting position for the origin of the initial character. These coordinates are relative to the origin of the specified drawable.

string Specifies the address of type **XChar2b** that contains the character string of 16-bit characters.

length Specifies the number of characters in the string argument.

fxdrawstring16 draws 16-bit text characters.

This routine can generate the errors **BadDrawable**, **BadMatch**, **BadFont**, and **BadGC**.

For more information about the C language **Xlib** routine called by the **FXlib** binding routine **fxdrawstring16**, see the **XDrawString16** routine in "Xlib Functions."

¦ Copyright IBM Corp. 1985, 1991

3.5.107 fxdrawtext

external fxdrawtext
integer*4 *display*
integer*4 *drawable, gc*
integer*4 *x, y, items, nitems*

call fxdrawtext(*display, drawable, gc, x, y, items, nitems*)

display Specifies the connection to the X Server.

drawable Specifies the drawable.

gc Specifies address of type **GC** that has the graphics context.

x, y Specify the *x* and *y* coordinates that define the baseline starting position for the origin of the initial character. These coordinates are relative to the origin of the specified drawable.

items Specifies the starting address of a list of addresses of type **XTextItem** that contain the text items.

nitems Specifies the number of text items in the array.

fxdrawtext draws 8-bit polytext characters.

For more information about the C language **Xlib** routine called by the **FXlib** binding routine **fxdrawtext**, see the **XDrawText** routine in "Xlib Functions."

¦ Copyright IBM Corp. 1985, 1991

3.5.108 fxdrawtext16

external fxdrawtext16
integer*4 *display*
integer*4 *drawable, gc*
integer*4 *x, y, items, nitems*

call fxdrawtext(*display, drawable, gc, x, y, items, nitems*)

display Specifies the connection to the X Server.

drawable Specifies the drawable.

gc Specifies an address of type **GC** that has the graphics context.

x, y Specify the *x* and *y* coordinates that define the baseline starting position for the origin of the initial character. These coordinates are relative to the origin of the specified drawable.

items Specifies the starting address of a list of addresses of type **XTextItem** that contain the text items.

nitems Specifies the number of text items in the array.

fxdrawtext16 draws 16-bit polytext characters.

For more information about the C language **Xlib** routine called by the **FXlib** binding routine **fxdrawtext16**, see the **XDrawText16** routine in "Xlib Functions."

¦ Copyright IBM Corp. 1985, 1991

3.5.109 fxemptyregion

```
integer*4 fxemptyregion
external  fxemptyregion
integer*4 r
integer*4 rc

rc = fxemptyregion( r)
```

r Specifies the address of type **_XRegion** of the region structure.

fxemptyregion determines if a specified region is empty.

Return value *rc* contains a nonzero if the region is empty.

For more information about the C language **Xlib** routine called by the **FXlib** binding routine **fxemptyregion**, see the **XEmptyRegion** routine in "Xlib Functions."

¦ Copyright IBM Corp. 1985, 1991

3.5.110 fxenableaccesscontrol

external fxenableaccesscontrol
integer*4 *display*

call fxenableaccesscontrol(*display*)

display Specifies the connection to the X Server.

fxenableaccesscontrol enables access control.

This routine can generate the error **BadAccess**.

For more information about the C language **Xlib** routine called by the **FXlib** binding routine **fxenableaccesscontrol**, see the **XEnableAccessControl** routine in "Xlib Functions."

¦ Copyright IBM Corp. 1985, 1991

3.5.111 fxequalregion

```
integer*4 fxequalregion
external  fxequalregion
integer*4 r1, r2
integer*4 rc

rc = fxequalregion( r1, r2)
```

r1, r2 Specify the address of type **_XRegion** of the two region structures that determine if these region structures have the same offset, size, and shape.

fxequalregion determines if two regions have the same offset, size, and shape.

Return value *rc* contains a value of nonzero if the two regions are identical.

For more information about the C language **Xlib** routine called by the **FXlib** binding routine **fxequalregion**, see the **XEqualRegion** routine in "Xlib Functions."

¦ Copyright IBM Corp. 1985, 1991

3.5.112 fxeventmaskofscreen

```
integer*4  fxeventmaskofscreen
external   fxeventmaskofscreen
integer*4  screen
integer*4  rc

rc = fxeventmaskofscreen( screen)
```

screen Specifies the address of type **Screen** of the screen of the display.

fxeventmaskofscreen returns the initial root event mask for the screen specified.

Return value *rc* is the initial root event mask of the screen specified.

For more information about the C language **Xlib** routine called by the **FXlib** binding routine **fxeventmaskofscreen**, see the **XEventMaskOfScreen** routine in "Xlib Functions."

¦ Copyright IBM Corp. 1985, 1991

3.5.113 fxeventsqueued

```
integer*4 fxeventsqueued
external  fxeventsqueued
integer*4 display
integer*4 mode
integer*4 rc

rc = fxeventsqueued( display, mode)
```

display Specifies the connection to the X Server.

mode Specifies the mode.

fxeventsqueued checks the number of events in the queue.

The variable *mode* can be set to one of the following: **QueuedAlready**, **QueuedAfterFlush**, or **QueuedAfterReading**.

Return value *rc* is the number of events received from the server, but not yet removed from the event queue.

For more information about the C language **Xlib** routine called by the **FXlib** binding routine **fxeventsqueued**, see the **XEventsQueued** routine in "Xlib Functions."

¦ Copyright IBM Corp. 1985, 1991

3.5.114 fxfetchbuffer

integer*4 fxfetchbuffer
external fxfetchbuffer
integer*4 *display*
integer*4 *nbytesreturn, returnbuffer*
integer*4 *fetchbuffer*

fetchbuffer = **fxfetchbuffer**(*display, nbytesreturn, returnbuffer*)

display Specifies the connection to the X Server.

nbytesreturn Returns the number of bytes in the buffer.

returnbuffer Specifies which buffer the data is to be returned from.

fxfetchbuffer returns data from a specified cut buffer.

Return value *fetchbuffer* is the starting address of a list of bytes that was returned.

fxfetchbuffer can generate the error **BadValue**.

For more information about the C language **Xlib** routine called by the **FXlib** binding routine **fxfetchbuffer**, see the **XFetchBuffer** routine in "Xlib Functions."

¦ Copyright IBM Corp. 1985, 1991

3.5.115 fxfetchbytes

integer*4 fxfetchbytes
external fxfetchbytes
integer*4 *display*
integer*4 *nbytesreturn*
integer*4 *fetchbytes*

fetchbytes = **fxfetchbytes**(*display, nbytesreturn*)

display Specifies the connection to the X Server.

nbytesreturn Returns the number of bytes in the buffer. The value zero is returned if there is no data in the buffer.

fxfetchbytes returns data from cut buffer zero.

Return value *fetchbytes* is the starting address of the list of bytes fetched.

fxfetchbytes can generate the error **BadWindow**.

For more information about the C language **Xlib** routine called by the **FXlib** binding routine **fxfetchbytes**, see the **XFetchBytes** routine in "Xlib Functions."

¦ Copyright IBM Corp. 1985, 1991

3.5.116 fxfetchname

```
integer*4 fxfetchname
external fxfetchname
integer*4 display
integer*4 window
integer*4 windownamereturn
integer*4 rc

rc = fxfetchname( display, window, windownamereturn)
```

display	Specifies the connection to the X Server.
window	Specifies the ID of the window whose icon name is being set.
windownamereturn	Returns the address of the window name that is a null-terminated string.

fxfetchname specifies the name of a window.

This routine can generate the event error **BadWindow**.

For more information about the C language **Xlib** routine called by the **FXlib** binding routine **fxfetchname**, see the **XFetchName** routine in "Xlib Functions."

¦ Copyright IBM Corp. 1985, 1991

3.5.117 fxfillarc

external fxfillarc
integer*4 *display*
integer*4 *drawable, gc*
integer*4 *x, y, width, height, angle1, angle2*

call fxfillarc(*display, drawable, gc, x, y, width, height, angle1,*
** angle2*)

display Specifies the connection to the X Server.

drawable Specifies the drawable.

gc Specifies an address of type **GC** that has the graphics context.

x, y Specify the *x* and *y* coordinates that are relative to the origin of the drawable. These coordinates specify the upper-left corner of the rectangle.

width, height Specify the *width* and *height*. of the major and minor axes of the arc.

angle1 Specifies the start of the *arc* relative to the 3 o'clock position from the center in units of degrees multiplied by 64.

angle2 Specifies the path and extent of the *arc* relative to the start of the arc in units of degrees multiplied by 64.

fxfillarc fills a single arc in the specified drawable.

This routine can generate the errors **BadDrawable**, **BadMatch**, and **BadGC**.

For more information about the C language **Xlib** routine called by the **FXlib** binding routine **fxfillarc**, see the **XFillArc** routine in "Xlib Functions."

¦ Copyright IBM Corp. 1985, 1991

3.5.118 fxfillarcs

external fxfillarcs
integer*4 *display*
integer*4 *drawable, gc*
integer*4 *arcs, narcs*

call fxfillarcs(*display, drawable, gc, arcs, narcs*)

display Specifies the connection to the X Server.

drawable Specifies the drawable.

gc Specifies an address of type **GC** that has the graphics context.

arcs Specifies an address of type **XArc** of an array of arcs. The client must allocate the arc list and must free the arc list when it is no longer needed.

narcs Specifies the number of arcs in the array.

fxfillarcs fills multiple arcs in the specified drawable.

This routine can generate the errors **BadDrawable**, **BadMatch**, and **BadGC**.

For more information about the C language **Xlib** routine called by the **FXlib** binding routine **fxfillarcs**, see the **XFillArcs** routine in "Xlib Functions."

¦ Copyright IBM Corp. 1985, 1991

3.5.119 fxfillpolygon

```
external  fxfillpolygon
integer*4 display
integer*4 drawable, gc
integer*4 points, npoints, shape, mode

call fxfillpolygon( display, drawable, gc, points, npoints, shape,
*                   mode)
```

display Specifies the connection to the X Server.

drawable Specifies the drawable.

gc Specifies an address of type **GC** that has the graphics context.

points Specifies an address of type **XPoint** of an array of points. The client must allocate the points list and must free the points list when it is no longer needed.

npoints Specifies the number of points in the array.

shape Specifies an argument that helps the server improve performance.

mode Specifies the coordinate mode.

fxfillpolygon fills a single polygon area in the specified drawable.

The variable *shape* can be set to **Complex**, **Convex**, or **Nonconvex**.

The variable *mode* can be set to **CoordModeOrigin**, which treats a coordinate as related to the origin, or to **CoordModePrevious**, which treats all coordinates after the first coordinate as relative to the previous point.

This routine can generate the errors **BadDrawable**, **BadMatch**, **BadValue**, and **BadGC**.

For more information about the C language **Xlib** routine called by the **FXlib** binding routine **fxfillpolygon**, see the **XFillPolygon** routine in "Xlib Functions."

¦ Copyright IBM Corp. 1985, 1991

3.5.120 fxfillrectangle

external fxfillrectangle
integer*4 *display*
integer*4 *drawable, gc*
integer*4 *x, y, width, height*

call fxfillrectangle(*display, drawable, gc, x, y, width, height*)

display Specifies the connection to the X Server.

drawable Specifies the drawable.

gc Specifies an address of type **GC** that has the graphics context.

x, y Specify the *x* and *y* coordinates of the *arc*. These coordinates are relative to the origin of the drawable. These coordinates specify the upper-left corner of the rectangle.

width, height Specify the *width* and *height* of the rectangle to be filled.

fxfillrectangle fills a single rectangular area in the specified drawable.

This routine can generate the errors **BadDrawable**, **BadMatch**, and **BadGC**.

For more information about the C language **Xlib** routine called by the **FXlib** binding routine **fxfillrectangle**, see the **XFillRectangle** routine in "Xlib Functions."

¦ Copyright IBM Corp. 1985, 1991

3.5.121 fxfillrectangles

external fxfillrectangles
integer*4 *display*
integer*4 *drawable, gc*
integer*4 *rectangles, nrectangles*

call fxfillrectangles(*display, drawable, gc, rectangles, nrectangles*)

display Specifies the connection to the X Server.

drawable Specifies the drawable.

gc Specifies an address of type **GC** that has the graphics context.

rectangles Specifies an address of type **XRectangle** of an array of rectangles. The client must allocate the rectangle list and must free the rectangle list when it is no longer needed.

nrectangles Specifies the number of rectangles in the array.

fxfillrectangles fills multiple rectangular areas in a specified drawable.

This routine can generate the errors **BadDrawable**, **BadMatch**, and **BadGC**.

For more information about the C language **Xlib** routine called by the **FXlib** binding routine **fxfillrectangles**, see the **XFillRectangles** routine in "Xlib Functions."

¦ Copyright IBM Corp. 1985, 1991

3.5.122 fxfindcontext

```
integer*4 fxfindcontext
external  fxfindcontext
integer*4 display, window, context
integer*4 datareturn
integer*4 status

status = fxfindcontext( display, window, context, datareturn)
```

display Specifies the connection to the X Server.

window Specifies the window associated with the data.

context Specifies the context type to which the data belongs.

datareturn Returns the starting address of the data.

fxfindcontext gets the data associated with a window and type.

For more information about the C language **Xlib** routine called by the **FXlib** binding routine **fxfindcontext**, see the **XFindContext** routine in "Xlib Functions."

¦ Copyright IBM Corp. 1985, 1991

3.5.123 fxflush

external fxflush
integer*4 *display*

call fxflush(*display*)

display Specifies the connection to the X Server.

fxflush flushes the output buffer.

For more information about the C language **Xlib** routine called by the **FXlib** binding routine **fxflush**, see the **XFlush** routine in "Xlib Functions."

¦ Copyright IBM Corp. 1985, 1991

3.5.124 fxforcescreensaver

external fxforcescreensaver
integer*4 *display*
integer*4 *mode*

call fxforcescreensaver(*display*, *mode*)

display Specifies the connection to the X Server.

mode Specifies the mode to be applied.

fxforcescreensaver applies the specified mode to the screen saver.

The variable *mode* can be set to **ScreenSaverActive** or **ScreenSaverReset**.

This routine can generate the error **BadValue**.

For more information about the C language **Xlib** routine called by the **Fxlib** binding routine **fxforcescreensaver**, see the **XForceScreenSaver** routine in "Xlib Functions."

¦ Copyright IBM Corp. 1985, 1991

3.5.125 fxfree

external **fxfree**
integer*4 *data*

call fxfree(*data*)

data Specifies the address of the data area to free.

fxfree frees in-memory data that was created by an **Xlib** function.

For more information about the C language **Xlib** routine called by the **FXlib** binding routine **fxfree**, see the **XFree** routine in "Xlib Functions."

¦ Copyright IBM Corp. 1985, 1991

3.5.126 fxfreecolormap

external fxfreecolormap
integer*4 *display*
integer*4 *cmap*

call fxfreecolormap(*display, cmap*)

display Specifies the connection to the X Server.

cmap Specifies the colormap identification which is the colormap associated with the resource identification to be deleted.

fxfreecolormap deletes the association between the colormap resource identification and the colormap.

This routine can generate the error **BadColor**.

For more information about the C language **Xlib** routine called by the **FXlib** binding routine **fxfreecolormap**, see the **XFreeColormap** routine in "Xlib Functions."

¦ Copyright IBM Corp. 1985, 1991

3.5.127 fxfreecolors

external fxfreecolors
integer*4 *display*
integer*4 *cmap*
integer*4 *pixels, npixels, planes*

call fxfreecolors(*display, cmap, pixels, npixels, planes*)

display Specifies the connection to the X Server.

cmap Specifies the colormap ID.

pixels Specifies an array of pixel values. These pixel values map to the cells in the specified colormap.

npixels Specifies number of pixels.

planes Specifies the planes to be freed.

fxfreecolors frees colormap cells.

fxfreecolors can generate the errors **BadColor, BadAccess,** and **BadValue**. All specified pixels that are allocated by the client in the colormap are freed, even if one or more pixels produce an error. A **BadValue** error is generated if a specified pixel is not a valid index into the colormap. A **BadAccess** error is generated if a specified pixel is not allocated by the client (unallocated or is only allocated by another client). If more than one error is generated, the one reported is arbitrary.

For more information about the C language **Xlib** routine called by the **FXlib** binding routine **fxfreecolors**, see the **XFreeColors** routine in "Xlib Functions."

¦ Copyright IBM Corp. 1985, 1991

3.5.128 fxfreecursor

external fxfreecursor
integer*4 *display*
integer*4 *cursor*

call fxfreecursor(*display, cursor*)

display Specifies the connection to the X Server.

cursor Specifies the cursor ID.

fxfreecursor frees or destroys the specified cursor.

This routine can generate error **BadCursor**.

For more information about the C language **Xlib** routine called by the **FXlib** binding routine **fxfreecursor**, see the **XFreeCursor** routine in "Xlib Functions."

¦ Copyright IBM Corp. 1985, 1991

3.5.129 fxfreefont

external fxfreefont
integer*4 *display*
integer*4 *fontstruct*

call fxfreefont(*display, fontstruct*)

display Specifies the connection to the X Server.

fontstruct Specifies the address of type **XFontStruct** that contains the font structure information.

fxfreefont unloads the font and frees the storage used by the font structure allocated by **fxqueryfont** or **fxloadqueryfont** in a single operation.

This routine can generate the error **BadFont**.

For more information about the C language **Xlib** routine called by the **FXlib** binding routine **fxfreefont**, see the **XFreeFont** routine in "Xlib Functions."

¦ Copyright IBM Corp. 1985, 1991

3.5.130 fxfreefontinfo

external fxfreefontinfo
integer*4 *names*
integer*4 *freeinfo, actualcount*

call fxfreefontinfo(*names, freeinfo, actualcount*)

names Specifies the starting address of the list of addresses of font names returned by **fxlistfontswithinfo**.

freeinfo Specifies the starting address of a list of addresses of type **XFontStruct** that has the font information returned by **fxlistfontswithinfo**.

actualcount Specifies the actual number of matched font names returned by **fxlistfontswithinfo**.

fxfreefontinfo frees the font information array.

For more information about the C language **Xlib** routine called by the **FXlib** binding routine **fxfreefont**, see the **XFreeFontInfo** routine in "Xlib Functions."

¦ Copyright IBM Corp. 1985, 1991

3.5.131 fxfreefontnames

external fxfreefontnames
integer*4 *list*

call fxfreefontnames(*list*)

list Specifies the starting address of a list of addresses that contain font names (character strings) to be freed.

fxfreefontnames frees a font name array.

For more information about the C language **Xlib** routine called by the **FXlib** binding routine **fxfreefontnames**, see the **XFreeFontNames** routine in "Xlib Functions."

¦ Copyright IBM Corp. 1985, 1991

3.5.132 fxfreefontpath

external fxfreefontpath
integer*4 *list*

call fxfreefontpath(*list*)

list Specifies the starting address of a list of addresses that contain font paths (character strings) to be freed.

fxfreefontpath frees data returned by **fxgetfontpath**.

For more information about the C language **Xlib** routine called by the **FXlib** binding routine **fxfreefontpath**, see the **XFreeFontPath** routine in "Xlib Functions."

¦ Copyright IBM Corp. 1985, 1991

3.5.133 fxfreegc

external fxfreegc
integer*4 *display*
integer*4 *gc*

call fxfreegc(*display, gc*)

display Specifies the connection to the X Server.

gc Specifies address of type **GC** that has the graphics context.

fxfreegc frees the specified graphics context.

This routine can generate error **BadGC**.

For more information about the C language **Xlib** routine called by the **FXlib** binding routine **fxfreegc**, see the **XFreeGC** routine in "Xlib Functions."

¦ Copyright IBM Corp. 1985, 1991

3.5.134 fxfreememory

```
external fxfreememory
integer*4 startaddress

call fxfreememory( startaddress)
```

startaddress Specifies the starting address of a segment of memory to be freed.

fxfreememory frees a segment of memory beginning at *startaddress*.

¦ Copyright IBM Corp. 1985, 1991

3.5.135 fxfreemodifiermapping

```
external fxfreemodifiermap
integer*4 modmap

call fxfreemodifiermap( modmap)
```

modmap Specifies the address of type **XModifierKeymap** of the keymap modifier data structure.

fxfreemodifiermapping destroys a keymap modifier structure.

For more information about the C language **Xlib** routine called by the **FXlib** binding routine **fxfreemodifiermap**, see the **XFreeModifiermap** routine in "Xlib Functions."

¦ Copyright IBM Corp. 1985, 1991

3.5.136 fxfreepixmap

```
external  fxfreepixmap
integer*4 display
integer*4 pixmap

call fxfreepixmap( display, pixmap)
```

display Specifies the connection to the X Server.

pixmap Specifies the pixmap.

fxfreepixmap frees all storage associated with a specified pixmap.

This routine can generate the error **BadPixmap**.

For more information about the C language **Xlib** routine called by the **FXlib** binding routine **fxfreepixmap**, see the **XFreePixmap** routine in "Xlib Functions."

¦ Copyright IBM Corp. 1985, 1991

3.5.137 fxgcontextfromgc

```
integer*4 fxgcontextfromgc
external  fxgcontextfromgc
integer*4 gc
integer*4 gcontext

gcontext = fxgcontextfromgc( gc)
```

gc Specifies the address of type **GC** of the graphics context.

fxgcontextfromgc obtains the **GContext** for the specified **GC**.

Return value *gcontext* specifies the resource ID of the graphics context.

For more information about the C language **Xlib** routine called by the **FXlib** binding routine **fxgcontextfromgc**, see the **XGContextFromGC** routine in "Xlib Functions."

¦ Copyright IBM Corp. 1985, 1991

3.5.138 fxgeometry

```
integer*4 fxgeometry
external  fxgeometry
integer*4 display
integer*4 screen
character*256 position, default
integer*4 bwidth, fwidth, fheight
integer*4 xadder, yadder
integer*4 xreturn, yreturn
integer*4 widthreturn, heightreturn
integer*4 changemask

changemask = fxgeometry( display, screen, position, default,
*                                  bwidth, fwidth, fheight, xadder, yadder,
*                                  xreturn, yreturn, widthreturn, heightretu
```

display	Specifies the connection to the X Server.
screen	Specifies the screen.
position, default	Specify the geometry specifications.
bwidth	Specifies the width of the border.
fwidth, fheight	Specify the font *height* and *width* in pixels (increment size).
xadder, yadder	Specify additional interior padding needed in the window.
xreturn, yreturn	Return the *x* offset and *y* offset determined.
widthreturn, heightreturn	Return the *width* and *height* determined.

fxgeometry parses window geometry given and argument and a default position.

Return value *changemask* is a bit mask that indicates which geometry values were found in the specifications. It also indicates the sign of the *x* and *y* values.

For more information about the C language **Xlib** routine called by the **FXlib** binding routine **fxgeometry**, see the **XGeometry** routine in "Xlib Functions."

¦ Copyright IBM Corp. 1985, 1991

3.5.139 fxgetatomname

```
character*256 fxgetatomname
external      fxgetatomname
integer*4     display
integer*4     atom
character*256 name

name = fxgetatomname( display, atom)
```

display Specifies the connection to the X Server.

atom Specifies the atom associated with the string name to be returned.

fxgetatomname gets the name for the specified atom identifier.

This routine can generate the error **BadAtom**.

For more information about the C language **Xlib** routine called by the **FXlib** binding routine **fxgetatomname**, see the **XGetAtomName** routine in "Xlib Functions."

¦ Copyright IBM Corp. 1985, 1991

3.5.140 fxgetclasshint

```
integer*4 fxgetclasshint
external fxgetclasshint
integer*4 display
integer*4 window, classhintsreturn
integer*4 rc

rc = fxgetclasshint( display, window, classhintsreturn)
```

display Specifies the connection to the X Server.

window Specifies the window ID.

classhintsreturn Returns the address of type **XClassHint** of the class hints returned.

fxgetclasshint gets the class of a window.

Return value *rc* indicates the status of getting the class hints.

This routine can generate the error **BadWindow**.

For more information about the C language **Xlib** routine called by the **FXlib** binding routine **fxgetclasshint**, see the **XGetClassHint** routine in "Xlib Functions."

¦ Copyright IBM Corp. 1985, 1991

3.5.141 fxgetdefault

```
character*256 fxgetdefault
external  fxgetdefault
integer*4 display
character*256  program
character*256  option
character*256 getdef

getdef = fxgetdefault( display, program, option)
```

display Specifies the connection to the X Server.

program Specifies the program name for the X defaults. This is the program name with the program argument, usually *argv*.

option Specifies the *option* name.

fxgetdefault finds the fonts, colors, and other environment defaults favored by a particular user.

Return value *getdef* is the string of the name for the option specified.

For more information about the C language **Xlib** routine called by the **FXlib** binding routine **fxgetdefault**, see the **XGetDefault** routine in "Xlib Functions."

¦ Copyright IBM Corp. 1985, 1991

3.5.142 fxgeterrordatabasetext

```
external  fxgeterrordatabasetext
integer*4 display
character*256  name
character*256  message
character*256  defaultstring
character*256  bufferreturn
integer*4 length

call fxgeterrordatabasetext( display, name, message,
*                                    defaultstring, bufferreturn, length)
```

display Specifies the connection to the X Server.

name Specifies the name of the application.

message Specifies the type of the error message.

defaultstring Specifies the default error message.

bufferreturn Returns the error description.

length Specifies the size of the buffer.

fxgeterrordatabasetext obtains error messages from the error database.

For more information about the C language **Xlib** routine called by the **FXlib** binding routine **fxgeterrordatabasetext**, see the **XGetErrorDatabaseText** routine in "Xlib Functions."

¦ Copyright IBM Corp. 1985, 1991

3.5.143 fxgeterrortext

external fxgeterrortext
integer*4 *display*
integer*4 *code*
character*256 *bufferreturn*
integer*4 *length*

call fxgeterrortext(*display, code, bufferreturn, length*)

display Specifies the connection to the X Server.

code Specifies the error code.

bufferreturn Returns the error description.

length Specifies the size of the buffer.

fxgeterrortext obtains textual descriptions of the specified error code.

For more information about the C language **Xlib** routine called by the **FXlib** binding routine **fxgeterrortext**, see the **XGetErrorText** routine in "Xlib Functions."

¦ Copyright IBM Corp. 1985, 1991

3.5.144 fxgetfontpath

integer*4 fxgetfontpath
external fxgetfontpath
integer*4 *display*
integer*4 *npathsreturn*
integer*4 *path*

path = **fxgetfontpath**(*display, npathsreturn*)

display Specifies the connection to the X Server.

npathsreturn Returns the number of strings in the font path array.

fxgetfontpath gets the current font search path.

Return value *path* is a starting address to a list of addresses that contain character strings of font search paths.

For more information about the C language **Xlib** routine called by the **FXlib** binding routine **fxgetfontpath**, see the **XGetFontPath** routine in "Xlib Functions."

¦ Copyright IBM Corp. 1985, 1991

3.5.145 fxgetfontproperty

```
integer*4 fxgetfontproperty
external  fxgetfontproperty
integer*4 fontstruct
integer*4 atom, valuereturn
integer*4 rc

rc = fxgetfontproperty( fontstruct, atom, valuereturn)
```

fontstruct Specifies the address of type **XFontStruct** that contains the font structure information.

atom Specifies the atom associated with the string name to be returned.

valuereturn Specifies the value of the font property.

fxgetfontproperty returns the specified font property.

Return value *rc* is the status of the **fxgetfontproperty** function. It is a value of 1 if the atom is defined, and a value of zero if the atom is not defined.

For more information about the C language **Xlib** routine called by the **FXlib** binding routine **fxgetfontproperty**, see the **XGetFontProperty** routine in "Xlib Functions."

¦ Copyright IBM Corp. 1985, 1991

3.5.146 fxgetgeometry

```
integer*4 fxgetgeometry
external  fxgetgeometry
integer*4 display
integer*4 drawable
integer*4 rootreturn
integer*4 xreturn, yreturn
integer*4 widthreturn, heightreturn
integer*4 borderwidthreturn
integer*4 depthreturn
integer*4 status

status = fxgetgeometry( display, drawable, rootreturn, xreturn,
*                       yreturn, widthreturn, heightreturn,
*                       borderwidthreturn, depthreturn)
```

display	Specifies the connection to the X Server.
drawable	Specifies the drawable.
rootreturn	Returns the root window ID for the specified window.
xreturn, yreturn	Return the *x* and *y* coordinates that define the location of the *drawable*. For a window, these coordinates specify the upper-left, outer corner relative to the origin of the parent window. For pixmaps, these coordinates are always zero.
widthreturn, heightreturn	Return the width and height of the *drawable*. For a window, these dimensions specify the size of the inside of the window. These dimensions do not include the border of the window.
borderwidthreturn	Returns the width of the border in pixels. The function returns the width of the border only if the drawable is a window. If the drawable is a pixmap, it returns a zero.
depthreturn	Returns the depth of the pixmap in bits-per-pixel for the object.

fxgetgeometry obtains the current geometry of the drawable specified.

Return value *status* contains the status of **fxgetgeometry**.

fxgetgeometry can generate the error **BadDrawable**.

For more information about the C language **Xlib** routine called by the **FXlib** binding routine **fxgetgeometry**, see the **XGetGeometry** routine in "Xlib Functions."

¦ Copyright IBM Corp. 1985, 1991

3.5.147 fxgeticonname

integer*4 fxgeticonname
external fxgeticonname
integer*4 *display*
integer*4 *window*
integer*4 *iconnamereturn*
integer*4 *rc*

rc = **fxgeticonname**(*display, window, iconnamereturn*)

display Specifies the connection to the X Server.

window Specifies the ID of the window whose icon name is being set.

iconnamereturn Returns the address of the name to be displayed in the icon window. The name is a null-terminated string.

fxgeticonname gets the name a window requests displayed in its icon.

This routine can generate the error **BadWindow**.

For more information about the C language **Xlib** routine called by the **FXlib** binding routine **fxgeticonname**, see the **XGetIconName** routine in "Xlib Functions."

¦ Copyright IBM Corp. 1985, 1991

3.5.148 fxgeticonsizes

```
integer*4 fxgeticonsizes
external fxgeticonsizes
integer*4 display
integer*4 window, sizelistreturn, countreturn
integer*4 rc

rc = fxgeticonsizes( display, window, sizelistreturn, countreturn)
```

display Specifies the connection to the X Server.

window Specifies the window ID.

sizelistreturn Returns the starting address of type **XIconSize** of the icon size list.

countreturn Returns the number of items in the size list.

fxgeticonsizes returns the value of the icon sizes atom.

Return value *rc* indicates the status of getting the icon size.

This routine can generate the error **BadWindow**.

For more information about the C language **Xlib** routine called by the **FXlib** binding routine **fxgeticonsizes**, see the **XGetIconSizes** routine in "Xlib Functions."

¦ Copyright IBM Corp. 1985, 1991

3.5.149 fxgetimage

```
integer*4  fxgetimage
external  fxgetimage
integer*4 display
integer*4 drawable
integer*4 x, y, width, height
integer*4 planemask, format
integer*4 image

image = fxgetimage( display, drawable, x, y, width, height, planemask,
*                           format)
```

display Specifies the connection to the X Server.

drawable Specifies the drawable.

x, y Specify the *x* and *y* coordinates that define the upper-left corner of the rectangle. These coordinates are relative to the origin of the drawable.

width, height Specify the *width* and *height* of the subimage. These arguments define the dimensions of the rectangle.

planemask Specifies the plane mask.

format Specifies the format for the image.

fxgetimage returns the content of a rectangle in the specified drawable on the display. This function is intended specifically for rudimentary hardcopy support.

Return value *image* is the address of type **XImage** that has the image structure information.

The variable *format* can be set to **XYPixmap** or **ZPixmap**.

fxgetimage can generate the errors **BadDrawable**, **BadMatch**, and **BadValue**.

For more information about the C language **Xlib** routine called by the **FXlib** binding routine **fxgetimage**, see the **XGetImage** routine in "Xlib Functions."

¦ Copyright IBM Corp. 1985, 1991

3.5.150 fxgetinputfocus

external fxgetinputfocus
integer*4 *display*
integer*4 *focusreturn, reverttoreturn*

call fxgetinputfocus(*display, focusreturn, reverttoreturn*)

display Specifies the connection to the X Server.

focusreturn Returns the focus window ID, **PointerRoot** or **None**.

reverttoreturn Returns the current focus state.

fxgetinputfocus gets the input focus.

The variable *reverttoreturn* can return **RevertToParent**, **RevertToPointerRoot**, or **RevertToNone**.

For more information about the C language **Xlib** routine called by the **FXlib** binding routine **fxgetinputfocus**, see the **XGetInputFocus** routine in "Xlib Functions."

¦ Copyright IBM Corp. 1985, 1991

3.5.151 fxgetkeyboardcontrol

```
external fxgetkeyboardcontrol
integer*4 display
integer*4 valuesreturn

call fxgetkeyboardcontrol( display, valuesreturn)
```

display Specifies the connection to the X Server.

valuesreturn Returns the address of type **XKeyboardState** that has the current keyboard parameters in the specified data structure.

fxgetkeyboardcontrol obtains the current control values for the keyboard.

For more information about the C language **Xlib** routine called by the **FXlib** binding routine **fxgetkeyboardcontrol**, see the **XGetKeyboardControl** routine in "Xlib Functions."

¦ Copyright IBM Corp. 1985, 1991

3.5.152 fxgetkeyboardmapping

```
integer*4 fxgetkeyboardmapping
external fxgetkeyboardmapping
integer*4 display
integer*4 firstkeycodewanted
integer*4 keycodecount
integer*4 keysymsperkeycodereturn
integer*4 keysym

keysym = fxgetkeyboardmapping( display, firstkeycodewanted,
*                                       keycodecount, keysymsperkeycoder
```

display	Specifies the connection to the X Server.
firstkeycodewanted	Specifies the first keycode that is to be returned.
keycodecount	Specifies the number of keycodes that are to be returned.
keysymsperkeycodereturn	Returns the number of *keysyms* per keycode.

fxgetkeyboardmapping gets the symbols for the specified number of keycodes.

Return value *keysym* is the starting address of a list of keyboard symbols.

This routine can generate the error **BadValue**.

For more information about the C language **Xlib** routine called by the **FXlib** binding routine **fxgetkeyboardmapping**, see the **XGetKeyboardMapping** routine in "Xlib Functions."

¦ Copyright IBM Corp. 1985, 1991

3.5.153 fxgetmodifiermapping

```
integer*4 fxgetmodifiermapping
external fxgetmodifiermapping
integer*4 display
integer*4 modmap

modmap = fxgetmodifiermapping( display)
```

display Specifies the connection to the X Server.

fxgetmodifiermapping obtains the keycodes that are used as modifiers.

Return value *modmap* is an address of type **XModifierKeymap** to a newly created keymap modifier structure.

For more information about the C language **Xlib** routine called by the **FXlib** binding routine **fxgetmodifiermapping**, see the **XGetModifierMapping** routine in "Xlib Functions."

¦ Copyright IBM Corp. 1985, 1991

3.5.154 fxgetmotionevents

```
integer*4 fxgetmotionevents
external  fxgetmotionevents
integer*4 display
integer*4 window, start, stop, neventsreturn
integer*4 timecoord

 timecoord = fxgetmotionevents( display, window, start, stop,
*                                          neventsreturn)
```

display Specifies the connection to the X Server.

window Specifies the window ID for the window with the pointer motion events to retrieve.

start, stop Specify the time interval in which the events are returned from the motion history buffer. The time intervals can be a time stamp, which is expressed in milliseconds, or **CurrentTime**. If the stop time is in the future, it is equivalent to specifying **CurrentTime**.

neventsreturn Returns the number of events from the motion history buffer.

fxgetmotionevents returns motion events for a specified window and the number of these events in the motion history buffer.

Return value *timecoord* is an address of type **XTimeCoord**. The address contains a list of motion events.

fxgetmotionevents generates error **BadWindow**.

For more information about the C language **Xlib** routine called by the **FXlib** binding routine **fxgetmotionevents**, see the **XGetMotionEvents** routine in "Xlib Functions."

¦ Copyright IBM Corp. 1985, 1991

3.5.155 fxgetnormalhints

```
integer*4 fxgetnormalhints
external fxgetnormalhints
integer*4 display
integer*4 window
integer*4 hintsreturn
integer*4 rc

rc = fxgetnormalhints ( display, window, hintsreturn)
```

display Specifies the connection to the X Server.

window Specifies the window ID.

hintsreturn Returns an address of type **XSizeHints** that contains the sizing hints for a window in its normal state.

fxgetnormalhints returns the size hints for a window in its normal state.

Return value *rc* indicates the status of getting the normal hints.

This routine can generate the error **BadWindow**.

For more information about the C language **Xlib** routine called by the **FXlib** binding routine **fxgetnormalhints**, see the **XGetNormalHints** routine in "Xlib Functions."

¦ Copyright IBM Corp. 1985, 1991

3.5.156 fxgetpixel

```
integer*4 fxgetpixel
external  fxgetpixel
integer*4 ximage, x, y
integer*4 pixel

pixel = fxgetpixel( ximage, x, y)
```

ximage Specifies the address of type **XImage** of the image structure.

x, y Specify the *x* and *y* coordinates relative to the origin of the image.

fxgetpixel obtains a pixel value in an image.

Return value *pixel* is the value in the image of the coordinates specified.

For more information about the C language **Xlib** routine called by the **FXlib** binding routine **fxgetpixel**, see the **XGetPixel** routine in "Xlib Functions."

¦ Copyright IBM Corp. 1985, 1991

3.5.157 fxgetpointercontrol

```
external fxgetpointercontrol
integer*4 display
integer*4 accelnumeratorreturn
integer*4 acceldenominatorreturn, thresholdreturn

call fxgetpointercontrol( display, accelnumeratorreturn,
*                              acceldenominatorreturn, thresholdreturn)
```

display	Specifies the connection to the X Server.
accelnumeratorreturn	Returns the numerator for the acceleration multiplier.
acceldenominatorreturn	Returns the denominator for the acceleration multiplier.
thresholdreturn	Returns the acceleration threshold.

fxgetpointercontrol gets the current parameters set for the pointer.

For more information about the C language **Xlib** routine called by the **FXlib** binding routine **fxgetpointercontrol**, see the **XGetPointerControl** routine in "Xlib Functions."

¦ Copyright IBM Corp. 1985, 1991

3.5.158 fxgetpointermapping

```
integer*4 fxgetpointermapping
external fxgetpointermapping
integer*4 display
integer*4 map
integer*4 nmap
integer*4 rc

rc = fxgetpointermapping( display, map, nmap)
```

display Specifies the connection to the X Server.

map Specifies the starting address of a list of the current pointer mapping one byte long.

nmap Specifies the number of items in mapping list.

fxgetpointermapping obtains the pointer mapping.

For more information about the C language **Xlib** routine called by the **FXlib** binding routine **fxgetpointermapping**, see the **XGetPointerMapping** routine in "Xlib Functions."

¦ Copyright IBM Corp. 1985, 1991

3.5.159 fxgetscreensaver

external fxgetscreensaver
integer*4 *display*
integer*4 *timeoutreturn, intervalreturn*
integer*4 *preferblankingreturn, allowexposuresreturn*

call fxgetscreensaver(*display, timeoutreturn, intervalreturn,*
preferblankingreturn, allowexposuresreturn)

display	Specifies the connection to the X Server.
timeoutreturn	Returns the timeout, in minutes, until the screen saver turns on.
intervalreturn	Returns the interval between screen saver invocations.
preferblankingreturn	Returns the current screen blanking preference.
allowexposuresreturn	Returns the current screen save control value.

fxgetscreensaver gets the current screen saver values.

The variable *preferblankingreturn* can be set to **DontPreferBlanking**, **PreferBlanking**, or **DefaultBlanking**.

The variable *allowexposuresreturn* can be set to **DontAllowExposures**, **AllowExposures**, or **DefaultExposures**.

For more information about the C language **Xlib** routine called by the **FXlib** binding routine **fxgetscreensaver**, see the **XGetScreenSaver** routine in "Xlib Functions."

¦ Copyright IBM Corp. 1985, 1991

3.5.160 fxgetselectionowner

integer*4 fxgetselectionowner
external fxgetselectionowner
integer*4 *display*
integer*4 *selection*
integer*4 *id*

id = **fxgetselectionowner**(*display, selection*)

display Specifies the connection to the X Server.

selection Specifies the selection atom whose owner is to be returned.

fxgetselectionowner returns the selection owner.

Return value *id* contains the window ID associated with the window that currently owns the specified selection. If no selection is specified, the function returns **None** which indicates that there is no owner for the selection.

fxgetselectionowner can generate the error **BadAtom**.

For more information about the C language **Xlib** routine called by the **FXlib** binding routine **fxgetselectionowner**, see the **XGetSelectionOwner** routine in "Xlib Functions."

¦ Copyright IBM Corp. 1985, 1991

3.5.161 fxgetsizehints

integer*4 fxgetsizehints
external fxgetsizehints
integer*4 *display*
integer*4 *window, hintsreturn, property*
integer*4 *rc*

rc = **fxgetsizehints**(*display, window, hintsreturn, property*)

display Specifies the connection to the X Server.

window Specifies the window ID.

hintsreturn Returns the address of type **XSizeHints** that contains the size hints.

property Specifies the property atom.

fxgetsizehints reads the value of any property type **XSizeHints**.

Return value *rc* indicates the status of getting the size hints.

This routine can generate the errors **BadWindow** and **BadAtom**.

For more information about the C language **Xlib** routine called by the **FXlib** binding routine **fxgetsizehints**, see the **XGetSizeHints** routine in "Xlib Functions."

¦ Copyright IBM Corp. 1985, 1991

3.5.162 fxgetstandardcolormap

```
integer*4 fxgetstandardcolormap
external fxgetstandardcolormap
integer*4 display
integer*4 window, cmapreturn, property
integer*4 rc

rc = fxgetstandardcolormap( display, window, cmapreturn, property)
```

display Specifies the connection to the X Server.

window Specifies the window ID.

cmapreturn Returns the address of the colormap associated with the specified atom.

property Specifies the property atom.

fxgetstandardcolormap gets the data structure associated with one of the described atoms.

This routine can generate the errors **BadWindow** and **BadAtom**.

For more information about the C language **Xlib** routine called by the **FXlib** binding routine **fxgetstandardcolormap**, see the **XGetStandardColormap** routine in "Xlib Functions."

¦ Copyright IBM Corp. 1985, 1991

3.5.163 fxgetstring

external fxgetstring
integer*4 fxgetstring
integer*4 *startaddress*
integer*4 *offset*
character*256 *string*

string = **fxgetstring**(*startaddress, offset*)

startaddress Specifies a starting address in memory used to determine an intermediate address, which contains the address of the desired string.

offset Specifies the offset in bytes from *startaddress* to the intermediate address.

fxgetstring gets a character string from memory. The address of the string is contained at an intermediate address. The intermediate address is located at *startaddress* plus *offset*.

Return value *string* contains the desired string.

¦ Copyright IBM Corp. 1985, 1991

3.5.164 fxgetstringaddress

```
external fxgetstringaddress
integer*4 fxgetstringaddress
integer*4 address
character*256 string

address = fxgetstringaddress( string)
```

string Specifies the string whose starting address is desired.

fxgetstringaddress gets the starting address of a string variable.

Return value *address* is the starting address of the string.

¦ Copyright IBM Corp. 1985, 1991

3.5.165 fxgetstringataddress

external fxgetstringataddress
character*256 fxgetstringataddress
integer*4 *valaddr*
character*256 *string*

string = **fxgetstringataddress**(*valaddr*)

valaddr Specifies the address at which the desired string is located.

fxgetstringataddress gets a character string from memory. The address of the string is contained in *valaddr*.

Return value *string* is the desired character string.

¦ Copyright IBM Corp. 1985, 1991

3.5.166 fxgetsubimage

```
integer*4  fxgetsubimage
external  fxgetsubimage
integer*4 display
integer*4 drawable
integer*4 x, y, width, height
integer*4 planemask, format
integer*4 destimage, destx, desty
integer*4 subimage

subimage = fxgetsubimage( display, drawable, x, y, width, height,
*                                  planemask, format, destimage, destx, dest
```

display Specifies the connection to the X Server.

drawable Specifies the drawable.

x, y Specify the *x* and *y* coordinates that define the upper-left corner of the rectangle. These coordinates are relative to the origin of the drawable.

width, height Specify the *width* and *height* of the subimage. These arguments define the dimensions of the rectangle.

planemask Specifies the plane mask.

format Specifies the format for the image.

destimage Specifies the destination image.

destx, desty Specify the *x* and *y* coordinates of the destination rectangle relative to its origin. These coordinates specify the upper-left corner of the destination rectangle and determine the location of the subimage within the destination image.

fxgetsubimage copies the contents of a rectangle in the specified drawable on the display to the specified location within a pre-existing image structure.

Return value *subimage* is the address of type **XImage** that has the image structure information.

The varible *format* can be set to **XYPixmap** or **ZPixmap**.

fxgetsubimage can generate the errors **BadDrawable**, **BadMatch**, **BadGC**, and **BadValue**.

For more information about the C language **Xlib** routine called by the **FXlib** binding routine **fxgetsubimage**, see the **XGetSubImage** routine in "Xlib Functions."

¦ Copyright IBM Corp. 1985, 1991

3.5.167 fxgettransientforhint

```
integer*4 fxgettransientforhint
external fxgettransientforhint
integer*4 display
integer*4 window, propwindowreturn
integer*4 rc

rc = fxgettransientforhint( display, window, propwindowreturn)
```

display	Specifies the connection to the X Server.
window	Specifies the window ID.
propwindowreturn	Returns the **WM_TRANSIENT_FOR** property for the specified window.

fxgettransientforhint gets the **WM_TRANSIENT_FOR** value for a window.

Return value *rc* indicates the status of the **WM_TRANSIENT_FOR** property for a specified window.

This routine can generate the error **BadWindow**.

For more information about the C language **Xlib** routine called by the **FXlib** binding routine **fxgettransientforhint**, see the **XGetTransientForHint** routine in "Xlib Functions."

¦ Copyright IBM Corp. 1985, 1991

3.5.168 fxgetvalue

```
external fxgetvalue
integer*4 fxgetvalue
integer*4 startaddress
integer*4 offset
integer*4 length
integer*4 returnvalue

returnvalue = fxgetvalue( startaddress, offset, length)
```

startaddress Specifies the starting address in memory used to determine the address for retrieving the data.

offset Specifies the offset from the starting address in bytes.

length Specifies the length of the data in bytes (1, 2, or 4).

fxgetvalue gets a 1, 2, or 4 byte value from memory at the address given by *startaddress* plus *offset*.

Return value *returnvalue* contains the value of the 1, 2, or 4 byte area in memory.

¦ Copyright IBM Corp. 1985, 1991

3.5.169 fxgetvisualinfo

integer*4 fxgetvisualinfo
external fxgetvisualinfo
integer*4 *display, infomask*
integer*4 *vinfotemplate*
integer*4 *nitemsreturn*
integer*4 *visualinfo*

visualinfo = **fxgetvisualinfo**(*display, infomask, vinfotemplate, nitemsreturn*)

display Specifies the connection to the X Server.

vinfomask Specifies the visual mask value.

vinfotemplate Specifies the address of type **XVisualInfo** that contains the visual attributes to be used in matching the visual structures.

nitemsreturn Returns the number of matching visual structures.

fxgetvisualinfo obtains a list of visual information structures that match a specified template.

Return value *visualinfo* is the starting address of type **XVisualInfo** of a list of Visual information structures. If no visual structures match the template using the specified *infomask*, **NULL** is returned. Use **fxfree** to free the data returned by this function.

For more information about the C language **Xlib** routine called by the **FXlib** binding routine **fxgetvisualinfo**, see the **XGetVisualInfo** routine in "Xlib Functions."

¦ Copyright IBM Corp. 1985, 1991

3.5.170 fxgetwindowattributes

```
integer*4 fxgetwindowattributes
external  fxgetwindowattributes
integer*4 display
integer*4 window
integer*4 windowattributesreturn
integer*4 status

status = fxgetwindowattributes( display, window, windowattributesreturn)
```

display	Specifies the connection to the X Server.
window	Specifies the window ID for the window whose current attributes are to be obtained.
windowattributesreturn	Returns the address of **XWindowAttributes** which has the specified window's attributes.

fxgetwindowattributes obtains the current attributes for a specified window.

Return value *status* contains the status of **fxgetwindowattributes**.

fxgetwindowattributes can generate the error **BadWindow**.

For more information about the C language **Xlib** routine called by the **FXlib** binding routine **fxgetwindowattributes**, see the **XGetWindowAttributes** routine in "Xlib Functions."

¦ Copyright IBM Corp. 1985, 1991

3.5.171 fxgetwindowproperty

```
integer*4 fxgetwindowproperty
external  fxgetwindowproperty
integer*4 display
integer*4 window
integer*4 property
integer*4 longoffset, longlength
integer*4 delete, reqtype, actualtypereturn
integer*4 actualformatreturn, nitemsreturn
integer*4 bytesafterreturn, propreturn
integer*4 rc

rc = fxgetwindowproperty( display, window, property, longoffset,
*                                  longlength, delete, reqtype, actualtypere
*                                  actualformatreturn, nitemsreturn,
*                                  bytesafterreturn, propreturn)
```

display	Specifies the connection to the X Server.
window	Specifies window ID for the window whose atom type and property format are to be obtained.
property	Specifies the property atom.
longoffset	Specifies the offset in 32-bit quantities for the property where data will be retrieved.
longlength	Specifies the length in 32-bit multiples of the data to be retrieved.
delete	Specifies a Boolean value that determines if the property is to be deleted from the window. This value should be set to **TRUE** or **FALSE**.
reqtype	Specifies the atom identifier associated with the property type. It should be set to an atom identifier or the constant **AnyPropertyType**.
actualtypereturn	Returns the atom identifier that defines the actual type of the property.
actualformatreturn	Returns the actual format of the property.
nitemsreturn	Returns the actual number of 8-bit, 16-bit, or 32-bit items transferred.
bytesafterreturn	Returns the number of bytes remaining to be read in the property if a partial read was performed.
propreturn	Returns the address of the data in the specified format.

fxgetwindowproperty obtains the atom type and property format for a specified window.

Return value *rc* contains the completion status of the call.

fxgetwindowproperty can generate the errors **BadAtom**, **BadValue**, or **BadWindow**.

¦ Copyright IBM Corp. 1985, 1991

For more information about the C language **Xlib** routine called by the **FXlib** binding routine **fxgetwindowproperty**, see the **XGetWindowProperty** routine in "Xlib Functions."

¦ Copyright IBM Corp. 1985, 1991

3.5.172 fxgetwmhints

integer*4 fxgetwmhints
external fxgetwmhints
integer*4 *display*
integer*4 *window*
integer*4 *wmhints*

wmhints = **fxgetwmhints**(*display, window*)

display Specifies the connection to the X Server.

window Specifies the window ID.

fxgetwmhints reads the value of the window manager hints atom.

Return value *wmhints* contains the window manager hints.

This routine can generate the errors **BadWindow** and **BadAlloc**.

For more information about the C language **Xlib** routine called by the **FXlib** binding routine **fxgetwmhints**, see the **XGetWMHints** routine in "Xlib Functions."

¦ Copyright IBM Corp. 1985, 1991

3.5.173 fxgetzoomhints

integer*4 fxgetzoomhints
external fxgetzoomhints
integer*4 *display*
integer*4 *window, zhintsreturn*
integer*4 *rc*

rc = **fxgetzoomhints**(*display, window, zhintsreturn*)

display Specifies the connection to the X Server.

window Specifies the window ID.

zhintsreturn Specifies an address of type **XSizeHints** that contains the zoom hints.

fxgetzoomhints returns the size hints for a window in its zoomed state.

Return value *rc* indicates the status of getting the zoom hints.

This routine can generate the event error **BadWindow**.

For more information about the C language **Xlib** routine called by the **FXlib** binding routine **fxgetzoomhints**, see the **XGetZoomHints** routine in "Xlib Functions."

¦ Copyright IBM Corp. 1985, 1991

3.5.174 fxgrabbutton

external fxgrabbutton
integer*4 *display*
integer*4 *buttongrab, modifiers, grabwindow*
integer*4 *ownerevents, eventmask, pointermode*
integer*4 *keyboardmode, confineto, cursor*

call fxgrabbutton(*display, buttongrab, modifiers, grabwindow,*
** ownerevents, eventmask, pointermode,*
** keyboardmode, confineto, cursor*)

display Specifies the connection to the X Server.

buttongrab Specifies the pointer button that is to be grabbed when the specified modifier keys are down. This variable can be set to **AnyButton**, which is equivalent to issuing the grab request for all possible buttons.

modifiers Specifies the set of keymasks. This is a bitwise inclusive OR of valid keymask bits.

grabwindow Specifies the window ID of the window to be grabbed.

ownerevents Specifies how the pointer events are to be reported.

eventmask Specifies which pointer events are reported to the client. This is a bitwise inclusive OR of valid pointer event mask bits.

pointermode Controls further processing of pointer events.

keyboardmode Controls further processing of keyboard events.

confineto Specifies the destination window for confining the pointer. Set to **None** if the pointer is not to be confined.

cursor Specifies the cursor that is to be displayed during the grab.

fxgrabbutton grabs a mouse (or pointer) button.

The variable *modifiers* can be set to include the following valid keymask bits:

ShiftMask, **LockMask**,

ControlMask, **ModMask**,

Mod2Mask, **Mod3Mask**,

Mod4Mask, **Mod5Mask**.

modifiers can also be set to **AnyModifier**, which is equivalent to issuing the grab request for all possible modifier combinations, including the combination of no modifiers.

The variable *ownerevents* can be set to **TRUE** if the pointer events are to be reported normally. Or, it can be set to **FALSE** if the pointer events

¦ Copyright IBM Corp. 1985, 1991

are to be selected by the event mask, with respect to the grab window.

eventmask can be set to one of the following valid pointer event-mask bits:

ButtonPressMask, **ButtonReleaseMask**,

EnterWindowMask, **LeaveWindowMask**,

PointerMotionMask, **PointerMotionHintMask**,

Button1MotionMask, **Button2MotionMask**,

Button3MotionMask, **Button4MotionMask**,

Button5MotionMask, **ButtonMotionMask**,

KeymapStateMask.

The variables *pointermode* and *keyboardmode* can be set to **GrabModeSync** or **GrabModeAsync**.

This routine can generate the errors **BadWindow**, **BadCursor**, **BadAccess**, and **BadValue**.

For more information about the C language **Xlib** routine called by the **FXlib** binding routine **fxgrabbutton**, see the **XGrabButton** routine in "Xlib Functions."

¦ Copyright IBM Corp. 1985, 1991

3.5.175 fxgrabkey

external fxgrabkey
integer*4 *display*
integer*4 *keycode, modifiers, grabwindow*
integer*4 *ownerevents, pointermode, keyboardmode*

call fxgrabkey(*display, keycode, modifiers, grabwindow,*
** ownerevents, pointermode, keyboardmode*)

display Specifies the connection to the X Server.

keycode Specifies the keycode which maps to the specific key to be grabbed. This variable can be the keycode or **AnyKey**.

modifiers Specifies the set of keymasks which is a bitwise inclusive OR of valid keymask bits.

grabwindow Specifies the window ID of the window associated with the keyboard to be grabbed.

ownerevents Specifies a Boolean value. It can be set to **TRUE** or **FALSE**.

pointermode Controls further processing of pointer events.

keyboardmode Controls further processing of keyboard events.

fxgrabkey passively grabs a single key of the keyboard.

The variable *modifiers* can be set to include the following valid keymask bits:

ShiftMask, **LockMask**,

ControlMask, **ModMask**,

Mod2Mask, **Mod3Mask**,

Mod4Mask, **Mod5Mask**.

modifiers can also be set to **AnyModifier**, which is equivalent to issuing the grab request for all possible modifier combinations, including the combination of no modifiers.

The variables *pointermode* and *keyboardmode* can be set to **GrabModeSync** or **GrabModeAsync**.

This routine can generate the errors **BadWindow**, **BadAccess**, and **BadValue**.

For more information about the C language **Xlib** routine called by the **FXlib** binding routine **fxgrabkey**, see the **XGrabKey** routine in "Xlib Functions."

¦ Copyright IBM Corp. 1985, 1991

3.5.176 fxgrabkeyboard

```
integer*4 fxgrabkeyboard
external fxgrabkeyboard
integer*4 display
integer*4 grabwindow, ownerevents
integer*4 pointermode, keyboardmode
integer*4 time
integer*4 rc
rc = fxgrabkeyboard( display, grabwindow, ownerevents,
*                              pointermode, keyboardmode, time)
```

display Specifies the connection to the X Server.

grabwindow Specifies the window ID of the window associated with the keyboard to be grabbed.

ownerevents Specifies a Boolean value. This variable can be set to **TRUE** or **FALSE**.

pointermode Controls further processing of pointer events.

keyboardmode Controls further processing of keyboard events.

time Specifies the time in time stamp, which can be expressed in milliseconds or **CurrentTime**.

fxgrabkeyboard grabs the keyboard.

The variables *pointermode* and *keyboardmode* can be set to **GrabModeSync** or **GrabModeAsync**.

Return value *rc* indicates the status of the grab.

fxgrabkeyboard can generate the errors **BadWindow** and **BadValue**.

For more information about the C language **Xlib** routine called by the **FXlib** binding routine **fxgrabkeyboard**, see the **XGrabKeyboard** routine in "Xlib Functions."

¦ Copyright IBM Corp. 1985, 1991

3.5.177 fxgrabpointer

```
integer*4 fxgrabpointer
external fxgrabpointer
integer*4 display
integer*4 grabwindow, ownerevents, eventmask
integer*4 pointermode, keyboardmode
integer*4 confineto, cursor, time
integer*4 rc

rc = fxgrabpointer( display, grabwindow, ownerevents,
*                           eventmask, pointermode, keyboardmode,
*                           confineto, cursor, time)
```

display Specifies the connection to the X Server.

grabwindow Specifies the window ID of the window relative to which events are reported while it is grabbed.

ownerevents Specifies how the pointer events are to be reported.

eventmask Specifies which pointer events are reported to the client. This is a bitwise inclusive OR of valid pointer event mask bits.

pointermode Controls further processing of pointer events.

keyboardmode Controls further processing of keyboard events.

confineto Specifies the destination window for confining the pointer. Use **None** if the pointer is not to be confined.

cursor Specifies the cursor that is to be displayed during the grab.

time Specifies the time in time stamp, which can be expressed in milliseconds or **CurrentTime**.

fxgrabpointer grabs the pointer.

The variable *ownerevents* can be set to **TRUE** if the pointer events are to be reported normally. Or, it can be set to **FALSE** if the pointer events are to be selected by the event mask with respect to the grab window.

The variable *eventmask* can be one of the following valid pointer event mask bits:

ButtonPressMask, **ButtonReleaseMask**,

EnterWindowMask, **LeaveWindowMask**,

PointerMotionMask, **PointerMotionHintMask**,

Button1MotionMask, **Button2MotionMask**,

Button3MotionMask, **Button4MotionMask**,

Button5MotionMask, **ButtonMotionMask**,

¦ Copyright IBM Corp. 1985, 1991

KeymapStateMask.

The variables *pointermode* and *keyboardmode* can be set to **GrabModeSync** or **GrabModeAsync**.

Return value *rc* indicates the status of the grab.

This routine can generate the errors **BadWindow**, **BadCursor**, and **BadValue**.

For more information about the C language **Xlib** routine called by the **FXlib** binding routine **fxgrabpointer**, see the **XGrabPointer** routine in "Xlib Functions."

¦ Copyright IBM Corp. 1985, 1991

3.5.178 fxgrabserver

external fxgrabserver
integer*4 *display*

call fxgrabserver(*display*)

display Specifies the connection to the X Server.

fxgrabserver grabs the server.

For more information about the C language **Xlib** routine called by the **FXlib** binding routine **fxgrabserver**, see the **XGrabServer** routine in "Xlib Functions."

¦ Copyright IBM Corp. 1985, 1991

3.5.179 fxheightmmofscreen

```
integer*4  fxheightmmofscreen
external   fxheightmmofscreen
integer*4  screen
integer*4  rc

rc = fxheightmmofscreen( screen)
```

screen Specifies the address of type **Screen** of the screen of the display.

fxheightmmofscreen returns the height in millimeters of the screen specified.

Return value *rc* is the height in millimeters of the screen specified.

For more information about the C language **Xlib** routine called by the **FXlib** binding routine **fxheightmmofscreen**, see the **XHeightMMOfScreen** routine in "Xlib Functions."

¦ Copyright IBM Corp. 1985, 1991

3.5.180 fxheightofscreen

```
integer*4  fxheightofscreen
external   fxheightofscreen
integer*4  screen
integer*4  rc

rc = fxheightofscreen( screen)
```

screen Specifies the address of type **Screen** of the screen of the display.

fxheightofscreen returns the height in pixels of the screen specified.

Return value *rc* is the height in pixels of the screen specified.

For more information about the C language **Xlib** routine called by the **FXlib** binding routine **fxheightofscreen**, see the **XHeightOfScreen** routine in "Xlib Functions."

¦ Copyright IBM Corp. 1985, 1991

3.5.181 fxifevent

external fxifevent
integer*4 *display*
integer*4 *eventreturn*
integer*4 *predicate*
character*256 *arg*

call fxifevent(*display, eventreturn, predicate, arg*)

display Specifies the connection to the X Server.

eventreturn Specifies the destination address of type **XEvent** in the client data area to copy the event structure from the event queue. The client must allocate the data area. The client must also free the data area when it is no longer needed.

predicate Specifies the procedure to call to determine if the next event in the queue matches the one specified by the event argument. The procedure must be declared **external**. Only the procedure name is passed to the binding routine.

arg Specifies the destination address of the user-supplied argument passed to the predicate procedure.

fxifevent flushes the output buffer and checks the event queue for the specified event. If the event matches, it removes the event from the queue.

For more information about the C language **Xlib** routine called by the **FXlib** binding routine **fxifevent**, see the **XIfEvent** routine in "Xlib Functions."

fxifevent requires a predicate procedure to pass as a parameter. The procedure determines if the event matches the one specified in the corresponding function. The predicate procedure is defined as follows:

external *functionname*
integer*4 *display*
integer*4 *event*
integer*4 *args*

call *functionname*(*display, event, args*)

functionname Specifies the user-supplied function name.

display Specifies the connection to the X Server.

event Specifies the address of type **XEvent** in the client data area.

args Specifies the arguments passed from the **fxifevent** function.

The *predicate* procedure is called once for each event in the queue until it finds a match between the event in the queue and the event specified by the corresponding function. After finding a match, the predicate procedure returns **TRUE**. If it does not find a match, it returns **FALSE**.

¦ Copyright IBM Corp. 1985, 1991

3.5.182 fximagebyteorder

```
integer*4  fximagebyteorder
external   fximagebyteorder
integer*4  display
integer*4  rc

rc = fximagebyteorder( display)
```

display Specifies the connection to the X Server.

fximagebyteorder obtains the byte order.

Return value *rc* is the value of the image byte order.

For more information about the C language **Xlib** routine called by the **FXlib** binding routine **fximagebyteorder**, see the **XImageByteOrder** routine in "Xlib Functions."

¦ Copyright IBM Corp. 1985, 1991

3.5.183 fxincrementaddress

external fxincrementaddress
integer*4 fxincrementaddress
integer*4 *startaddress*
integer*4 *numberbytes*
integer*4 *address*

address = **fxincrementaddress**(*startaddress*, *numberbytes*)

startaddress Specifies the starting address to be incremented.

numberbytes Specifies the number of bytes to increment the specified address.

fxincrementaddress increments an address by a specified number of bytes.

Return value *address* contains the address that is incremented.

¦ Copyright IBM Corp. 1985, 1991

3.5.184 fxinsertmodifiermapentry

```
integer*4 fxinsertmodifiermapentry
external fxinsertmodifiermapentry
integer*4 modmap
integer*4 keysyment
integer*4 modifier
integer*4 keymap

keymap = fxinsertmodifiermapentry( modmap, keysyment, modifier)
```

modmap Specifies the address of type **XModifierKeymap** of the keymap modifier data structure.

keysyment Specifies the *keysyms*.

modifier Specifies the modifier.

fxinsertmodifiermapentry adds a new entry to the **XModifierKeymap** data structure.

For more information about the C language **Xlib** routine called by the **FXlib** binding routine **fxinsertmodifiermapentry**, see the **XInsertModifiermapEntry** routine in "Xlib Functions."

¦ Copyright IBM Corp. 1985, 1991

3.5.185 fxinstallcolormap

external fxinstallcolormap
integer*4 *display*
integer*4 *cmap*

call fxinstallcolormap(*display*, *cmap*)

display Specifies the connection to the X Server.

cmap Specifies the colormap ID.

fxinstallcolormap installs a colormap.

This routine can generate the error **BadColor**.

For more information about the C language **Xlib** routine called by the **FXlib** binding routine **fxinstallcolormap**, see the **XInstallColormap** routine in "Xlib Functions."

¦ Copyright IBM Corp. 1985, 1991

3.5.186 fxinternatom

integer*4 fxinternatom
external fxinternatom
integer*4 *display*
character*256 *atomname*
integer*4 *onlyifexists*
integer*4 *id*

id = **fxinternatom**(*display, atomname, onlyifexists*)

display Specifies the connection to the X Server.

atomname Specifies the name associated with the atom to be returned.

onlyifexists Specifies a Boolean value that indicates whether **fxinternatom** creates the atom. This value should be set to **TRUE** or **FALSE**.

fxinternatom gets an atom identifier for a specified name.

Return value *id* contains a value of type **Atom**.

fxinternatom can generate the errors **BadAlloc** and **BadValue**.

For more information about the C language **Xlib** routine called by the **FXlib** binding routine **fxinternatom**, see the **XInternAtom** routine in "Xlib Functions."

¦ Copyright IBM Corp. 1985, 1991

3.5.187 fxintersectregion

external fxintersectregion
integer*4 *sra, srb, dr*

call fxintersectregion(*sra, srb, dr*)

sra, srb Specify the addresses of type **_XRegion** of the two region structures with which to perform the computation.

dr Specifies the address of type **_XRegion** of the resulting region structure.

fxintersectregion computes the intersection of two regions.

For more information about the C language **Xlib** routine called by the **FXlib** binding routine **fxintersectregion**, see the **XIntersectRegion** routine in "Xlib Functions."

¦ Copyright IBM Corp. 1985, 1991

3.5.188 fxkeycodetokeysym

```
integer*4 fxkeycodetokeysym
external  fxkeycodetokeysym
integer*4 display
integer*4 keycode, indexreturn
integer*4 keysym

keysym = fxkeycodetokeysym( display, keycode, indexreturn)
```

display Specifies the connection to the X Server.

keycode Specifies the keycode.

indexreturn Returns the element of the keycode vector.

fxkeycodetokeysym converts a keycode to a defined keysym.

Return value *keysym* is the value defined for the keycode specified and the element of the keycode vector.

For more information about the C language **Xlib** routine called by the **FXlib** binding routine **fxkeycodetokeysym**, see the **XKeycodeToKeysym** routine in "Xlib Functions."

¦ Copyright IBM Corp. 1985, 1991

3.5.189 fxkeysymtokeycode

```
integer*4 fxkeysymtokeycode
external  fxkeysymtokeycode
integer*4 display
integer*4 keysymkcode
integer*4 keycode

keycode = fxkeysymtokeycode( display, keysymkcode)
```

display Specifies the connection to the X Server.

keysymkcode Specifies the *keysym* to be converted.

fxkeysymtokeycode converts a *keysym* to the appropriate keycode.

Return value *keycode* is the value corresponding to the keysym specified.

For more information about the C language **Xlib** routine called by the **FXlib** binding routine **fxkeysymtokeycode**, see the **XKeysymToKeycode** routine in "Xlib Functions."

¦ Copyright IBM Corp. 1985, 1991

3.5.190 fxkeysymtostring

character*256 fxkeysymtostring
external fxkeysymtostring
integer*4 *keysymstr*
character*256 *keysymname*

keysymname = **fxkeysymtostring**(*keysymstr*)

keysymstr Specifies the keysym to be converted.

fxkeysymtostring converts a keysym code to the keysym name.

Return value *keysymname* contains the keysym name.

For more information about the C language **Xlib** routine called by the **FXlib** binding routine **fxkeysymtostring**, see the **XKeysymToString** routine in "Xlib Functions."

¦ Copyright IBM Corp. 1985, 1991

3.5.191 fxkillclient

external fxkillclient
integer*4 *display*
integer*4 *resource*

call fxkillclient(*display, resource*)

display Specifies the connection to the X Server.

resource Specifies any resource associated with the client to be destroyed. This variable can also be set to **AllTemporary**.

fxkillclient destroys a client.

This routine can generate the error **BadValue**.

For more information about the C language **Xlib** routine called by the **FXlib** binding routine **fxkillclient**, see the **XKillClient** routine in "Xlib Functions."

¦ Copyright IBM Corp. 1985, 1991

3.5.192 fxlastknownrequestprocessed

```
integer*4  fxlastknownrequestprocessed
external   fxlastknownrequestprocessed
integer*4  display
integer*4  rc

rc = fxlastknownrequestprocessed( display)
```

display Specifies the connection to the X Server.

fxlastknownrequestprocessed obtains the full serial number of the last request known by **Xlib** processed by the X Server.

Return value *rc* is the full serial number of the last known request.

For more information about the C language **Xlib** routine called by the **FXlib** binding routine **fxlastknownrequestprocessed**, see the **XLastKnownRequestProcessed** routine in "Xlib Functions."

¦ Copyright IBM Corp. 1985, 1991

3.5.193 fxlistfonts

```
integer*4 fxlistfonts
external  fxlistfonts
integer*4 display
character*256  pattern
integer*4 maxnames, actualcountreturn
integer*4 fontinfo
fontinfo = fxlistfonts( display, pattern, maxnames,
*                              actualcountreturn)
```

display Specifies the connection to the X Server.

pattern Specifies the string associated with the font names to be returned. You can specify:

- Any string
- An asterisk (*) that indicates a wild card on any number of characters
- A question mark (?) that indicates a wild card on a single character.

maxnames Specifies the maximum number of names that are to be in the returned list.

actualcountreturn Returns the actual number of matched font names.

fxlistfonts returns a list of the available font names.

Return value *fontinfo* is a starting address to a list of addresses that contain character strings of font names that match the *pattern* specification.

For more information about the C language **Xlib** routine called by the **FXlib** binding routine **fxlistfonts**, see the **XListFonts** routine in "Xlib Functions."

¦ Copyright IBM Corp. 1985, 1991

3.5.194 fxlistfontswithinfo

```
integer*4 fxlistfontswithinfo
external  fxlistfontswithinfo
integer*4 display
character*256  pattern
integer*4 maxnames, countreturn, inforeturn
integer*4 fontinfo

fontinfo = fxlistfontswithinfo( display, pattern, maxnames,
*                                    countreturn, inforeturn)
```

display Specifies the connection to the X Server.

pattern Specifies the string associated with the font names that is to be returned. You can specify:

Any string
An asterisk (*) that indicates a wild card on any number of characters
a question mark (?) that indicates a wild card on a single character.

maxnames Specifies the maximum number of names that are to be in the returned list.

countreturn Returns the actual number of matched font names.

inforeturn Returns the starting address of a list of addresses of type **XFontStruct** that has the font information for each font that matches the pattern specification.

fxlistfontswithinfo obtains the names and information about loaded fonts.

Return value *fontinfo* is a starting address of a list of addresses. At each address in the list are character strings of font names that match the pattern specification.

For more information about the C language **Xlib** routine called by the **FXlib** binding routine **fxlistfontswithinfo**, see the **XListFontsWithInfo** routine in "Xlib Functions."

¦ Copyright IBM Corp. 1985, 1991

3.5.195 fxlisthosts

```
integer*4 fxlisthosts
external fxlisthosts
integer*4 display, nhostsreturn, statereturn
integer*4 hostaddr

hostaddr = fxlisthosts( display, nhostsreturn, statereturn)
```

display Specifies the connection to the X Server.

nhostsreturn Returns the number of hosts currently in the access control list.

statereturn Returns the state of the access control.

fxlisthosts obtains a host list.

The variable *statereturn* can be set to **EnableAccess**, which enables host access control. Or, it can be set to **DisableAccess**, which disables host access control.

Return value *hostaddr* is the starting address of type **XHostAddress** for a list of each host in the access control list.

For more information about the C language **Xlib** routine called by the **FXlib** binding routine **fxlisthosts**, see the **XListHosts** routine in "Xlib Functions."

¦ Copyright IBM Corp. 1985, 1991

3.5.196 fxlistinstalledcolormaps

```
integer*4 fxlistinstalledcolormaps
external fxlistinstalledcolormaps
integer*4 display
integer*4 window, numreturn
integer*4 colormap

colormap = fxlistinstalledcolormaps( display, window, numreturn)
```

display Specifies the connection to the X Server.

window Specifies the window ID.

numreturn Returns the list of currently installed colormaps.

fxlistinstalledcolormaps provides a list of the currently installed colormaps.

Return value *colormap* is the starting address of the list of colormaps currently installed for the screen of the specified window.

This routine can generate the error **BadWindow**.

For more information about the C language **Xlib** routine called by the **FXlib** binding routine **fxlistinstalledcolormaps**, see the **XListInstalledColormaps** routine in "Xlib Functions."

¦ Copyright IBM Corp. 1985, 1991

3.5.197 fxlistproperties

```
integer*4 fxlistproperties
external  fxlistproperties
integer*4 display
integer*4 window
integer*4 numpropreturn

atom = fxlistproperties ( display, window, numpropreturn)
```

display Specifies the connection to the X Server.

window Specifies window ID for the window whose property list is to be obtained.

numpropreturn Returns the length of the properties array.

fxlistproperties obtains a property list for a specified window.

Return value *atom* contains the address of an array of atom properties that are defined for the specified window. Use **fxfree** to free the memory allocated by the **XListProperties**.

fxlistproperties can generate the error **BadWindow**.

For more information about the C language **Xlib** routine called by the **FXlib** binding routine **fxlistproperties**, see the **XListProperties** routine in "Xlib Functions."

¦ Copyright IBM Corp. 1985, 1991

3.5.198 fxloadfont

```
integer*4 fxloadfont
external  fxloadfont
integer*4 display
character*256  name
integer*4 font

font = fxloadfont( display, name)
```

display Specifies the connection to the X Server.

name Specifies the name of the font. This name is a null-terminated string. It is not case-sensitive.

fxloadfont loads the specified font.

This routine can generate the errors **BadAlloc** and **BadName**.

For more information about the C language **Xlib** routine called by the **FXlib** binding routine **fxloadfont**, see the **XLoadFont** routine in "Xlib Functions."

¦ Copyright IBM Corp. 1985, 1991

3.5.199 fxloadqueryfont

```
integer*4 fxloadqueryfont
external  fxloadqueryfont
integer*4 display
character*256  name
integer*4 font

font = fxloadqueryfont( display, name)
```

display Specifies the connection to the X Server.

name Specifies the name of the font. The name is a null-terminated string.

fxloadqueryfont performs an **fxloadfont** and **fxqueryfont** in a single operation.

Return value *font* is an address of type **XFontStruct** that has the font structure information.

This routine can generate the error **BadAlloc**.

For more information about the C language **Xlib** routine called by the **FXlib** binding routine **fxloadqueryfont**, see the **XLoadQueryFont** routine in "Xlib Functions."

¦ Copyright IBM Corp. 1985, 1991

3.5.200 fxlookupcolor

```
integer*4 fxlookupcolor
external  fxlookupcolor
integer*4 display
integer*4 cmap
character*256  colorname
integer*4 screendefreturn
integer*4 exactdefreturn
integer*4 status

status = fxlookupcolor( display, cmap, colorname,
*                              screendefreturn, exactdefreturn)
```

display Specifies the connection to the X Server.

cmap Specifies the colormap ID.

colorname Specifies the color-name string for the color definition structure returned, for example *red*.

screendefreturn Returns an address of type **XColor** which has the values used in the colormap.

exactdefreturn Returns an address of type **XColor**, which has the true pixel values of the closest color provided by the hardware for the color name specified.

fxlookupcolor looks up the name of a color.

The variable *status* contains zero if the color string specified is in the RGB database. This variable contains a non-zero if the color string specified is not in the RGB database.

fxlookupcolor can generate the errors **BadColor** and **BadName**.

For more information about the C language **Xlib** routine called by the **FXlib** binding routine **fxlookupcolor**, see the **XLookupColor** routine in "Xlib Functions."

¦ Copyright IBM Corp. 1985, 1991

3.5.201 fxlookupkeysym

integer*4 fxlookupkeysym
external fxlookupkeysym
integer*4 *eventkey*
integer*4 *index*
integer*4 *keysym*

keysym = **fxlookupkeysym**(*eventkey, index*)

eventkey Specifies the address of type **XKeyEvent** that contains the key event to be used.

index Specifies the *index* into the KeySyms table.

fxlookupkeysym looks up the Keysyms.

The variable *eventkey* can be set to **KeyPress** event or to **KeyRelease** event.

Return value *keysym* is the KeySym that corresponds to the keycode member in the event structure.

For more information about the C language **Xlib** routine called by the **FXlib** binding routine **fxlookupkeysym**, see the **XLookupKeysym** routine in "Xlib Functions."

¦ Copyright IBM Corp. 1985, 1991

3.5.202 fxlookupmapping

```
external fxlookupmapping
integer*4 fxlookupmapping
integer*4 event
integer*4 nbytes
integer*4 map

map = fxlookupmapping ( event, nbytes)
```

event Specifies the address of the character string mapped to this event.

nbytes Specifies the number of bytes in the character string or zero if no text is mapped to the event.

fxlookupmapping maps events to counted character strings.

The return value map contains the address of a character string, which must not be modified.

For more information about the C language **Xlib** routine called by the **FXlib** binding routine **fxlookupmapping**, see the **XLookupMapping** routine in "Xlib Functions."

¦ Copyright IBM Corp. 1985, 1991

3.5.203 fxlookupstring

```
integer*4 fxlookupstring
external  fxlookupstring
integer*4 eventstruct
character*256 bufferreturn
integer*4 bytesbuffer
integer*4 keysymreturn, statusreturn
integer*4 strlen

strlen = fxlookupstring( eventstruct, bufferreturn, bytesbuffer,
*                                keysymreturn, statusreturn)
```

eventstruct Specifies the address of type **XKeyEvent**.

bufferreturn Returns the translated characters.

bytesbuffer Specifies the length of the buffer. No more than *bytesbuffer* of translation is returned.

keysymreturn Returns the *keysym* computed from the event if this argument is not **NULL**.

statusreturn Specifies an address of type **XComposeStatus** that contains compose-key state information and allows compose-key processing to take place. Otherwise, it specifies **NULL**.

fxlookupstring maps a key event to an ASCII string.

The variable *eventstruct* can be set to **XKeyPressedEvent** or **XKeyReleasedEvent**.

Return value *strlen* is the length of the string in the tag buffer.

For more information about the C language **Xlib** routine called by the **FXlib** binding routine **fxlookupstring**, see the **XLookupString** routine in "Xlib Functions."

¦ Copyright IBM Corp. 1985, 1991

3.5.204 fxlowerwindow

external fxlowerwindow
integer*4 *display*
integer*4 *window*

call fxlowerwindow(*display, window*)

display Specifies the connection to the X Server.

window Specifies the window ID.

fxlowerwindow lowers a window so that it does not hide any sibling windows.

For more information about the C language **Xlib** routine called by the **FXlib** binding routine **fxlowerwindow**, see the **XLowerWindow** routine in "Xlib Functions."

¦ Copyright IBM Corp. 1985, 1991

3.5.205 fxmapraised

external fxmapraised
integer*4 *display*
integer*4 *window*

call fxmapraised(*display, window*)

display Specifies the connection to the X Server.

window Specifies the window ID.

fxmapraised maps and raises a window.

For more information about the C language **Xlib** routine called by the **FXlib** binding routine **fxmapraised**, see the **XMapRaised** routine in "Xlib Functions."

¦ Copyright IBM Corp. 1985, 1991

3.5.206 fxmapsubwindows

external fxmapsubwindows
integer*4 *display*
integer*4 *window*

call fxmapsubwindows(*display, window*)

display Specifies the connection to the X Server.

window Specifies the window ID.

fxmapsubwindows maps all subwindows for a specified window.

For more information about the C language **Xlib** routine called by the **FXlib** binding routine **fxmapsubwindows**, see the **XMapSubwindows** routine in "Xlib Functions."

¦ Copyright IBM Corp. 1985, 1991

3.5.207 fxmapwindow

external fxmapwindow
integer*4 *display*
integer*4 *window*

call fxmapwindow(*display, window*)

display Specifies the connection to the X Server.

window Specifies the window ID.

fxmapwindow maps the specified window.

For more information about the C language **Xlib** routine called by the **FXlib** binding routine **fxmapwindow**, see the **XMapWindow** routine in "Xlib Functions."

¦ Copyright IBM Corp. 1985, 1991

3.5.208 fxmaskevent

external fxmaskevent
integer*4 *display*
integer*4 *eventmask, eventreturn*

call fxmaskevent(*display, eventmask, eventreturn*)

display Specifies the connection to the X Server.

eventmask Specifies the event mask. This mask is the bitwise inclusive OR of one or more of the valid *event* mask bits.

eventreturn Specifies the destination address of type **XEvent** in the client data area to copy the event structure from the event queue. The client must allocate the data area. The client must also free the data area when it is no longer needed.

fxmaskevent removes the next event in the queue that matches an event mask.

For more information about the C language **Xlib** routine called by the **FXlib** binding routine **fxmaskevent**, see the **XMaskEvent** routine in "Xlib Functions."

¦ Copyright IBM Corp. 1985, 1991

3.5.209 fxmatchvisualinfo

```
integer*4 fxmatchvisualinfo
external  fxmatchvisualinfo
integer*4 display, screen
integer*4 depth, class
integer*4 vinforeturn
integer*4 status

status = fxmatchvisualinfo( display, screen, depth, class,
*                                   vinforeturn)
```

display Specifies the connection to the X Server.

screen Specifies the *screen*.

depth Specifies the *depth* of the screen.

class Specifies the *class* of the screen.

vinforeturn Returns the address of type **XVisualInfo** of a list of visual information structures that match the specified visual information.

fxmatchvisualinfo obtains the visual information that matches the specified depth and class of the screen.

If a visual is found, *status* contains **TRUE**; if not found, it contains **FALSE**.

For more information about the C language **Xlib** routine called by the **FXlib** binding routine **fxmatchvisualinfo**, see the **XMatchVisualInfo** routine in "Xlib Functions."

¦ Copyright IBM Corp. 1985, 1991

3.5.210 fxmaxcmapsofscreen

```
integer*4  fxmaxcmapsofscreen
external   fxmaxcmapsofscreen
integer*4  screen
integer*4  rc

rc = fxmaxcmapsofscreen( screen)
```

screen Specifies the address of type **Screen** of the screen of the display.

fxmaxcmapsofscreen returns the maximum number of colormaps supported by the screen specified.

Return value *rc* is the maximum number of colormaps supported by the screen specified.

For more information about the C language **Xlib** routine called by the **FXlib** binding routine **fxmaxcmapsofscreen**, see the **XMaxCmapsOfScreen** routine in "Xlib Functions."

¦ Copyright IBM Corp. 1985, 1991

3.5.211 fxmincmapsofscreen

```
integer*4  fxmincmapsofscreen
external   fxmincmapsofscreen
integer*4  screen
integer*4  rc

rc = fxmincmapsofscreen( screen)
```

screen Specifies the address of type **Screen** of the screen of the display.

fxmincmapsofscreen returns the minimum number of colormaps supported by the screen specified.

Return value *rc* is the minimum number of colormaps supported by the screen specified.

For more information about the C language **Xlib** routine called by the **FXlib** binding routine **fxmincmapsofscreen**, see the **XMinCmapsOfScreen** routine in "Xlib Functions."

¦ Copyright IBM Corp. 1985, 1991

3.5.212 fxmoveresizewindow

external fxmoveresizewindow
integer*4 *display*
integer*4 *window*
integer*4 *x*
integer*4 *y*
integer*4 *width*
integer*4 *height*

call fxmoveresizewindow(*display, window, x, y, width, height*)

display Specifies the connection to the X Server.

window Specifies the window ID.

x, y Specify the *x* and *y* coordinates. These coordinates define the new position of the window relative to its parent.

width, height Specify the *width* and *height*. These arguments define the interior size of the window.

fxmoveresizewindow changes the size and location of a window.

For more information about the C language **Xlib** routine called by the **FXlib** binding routine **fxmoveresizewindow**, see the **XMoveResizeWindow** routine in "Xlib Functions."

¦ Copyright IBM Corp. 1985, 1991

3.5.213 fxmovewindow

external fxmovewindow
integer*4 *display*
integer*4 *window*
integer*4 *x*
integer*4 *y*

call fxmovewindow(*display*, *window*, *x*, *y*)

display Specifies the connection to the X Server.

window Specifies the window ID.

x, y Specify the *x* and *y* coordinates. These coordinates define the new location of the top-left pixel of the window border, or the window itself if it has no border.

fxmovewindow moves a window without changing the size of the window.

For more information about the C language **Xlib** routine called by the **FXlib** binding routine **fxmovewindow**, see the **XMoveWindow** routine in "Xlib Functions."

¦ Copyright IBM Corp. 1985, 1991

3.5.214 fxnewmodifiermapping

```
integer*4 fxnewmodifiermapping
external fxnewmodifiermapping
integer*4 maxkeyspermod
integer*4 modmap

modmap = fxnewmodifiermapping ( maxkeyspermod)
```

maxkeyspermod Specifies the maximum number of keycodes assigned to any of the modifiers in the map.

fxnewmodifiermapping creates a new keymap modifier structure.

Return value *modmap* is an address of type **XModifierKeymap** to the new keymap modifier structure.

For more information about the C language **Xlib** routine called by the **FXlib** binding routine **fxnewmodifiermapping**, see the **XNewModifierMapping** routine in "Xlib Functions."

¦ Copyright IBM Corp. 1985, 1991

3.5.215 fxnextevent

external fxnextevent
integer*4 *display*
integer*4 *eventreturn*

call fxnextevent(*display, eventreturn*)

display Specifies the connection to the X Server.

eventreturn Specifies the destination address of type **XEvent** in the client data area to copy the event structure from the event queue. The client must allocate the data area. The client must also free the data area when it is no longer needed.

fxnextevent flushes the output buffer, copies the next event, and then removes the event from the event queue.

For more information about the C language **Xlib** routine called by the **FXlib** binding routine **fxnextevent**, see the **XNextEvent** routine in "Xlib Functions."

¦ Copyright IBM Corp. 1985, 1991

3.5.216 fxnextrequest

```
integer*4  fxnextrequest
external   fxnextrequest
integer*4  display
integer*4  rc

rc = fxnextrequest( display)
```

display Specifies the connection to the X Server.

fxnextrequest obtains the full serial number to use for the next request.

Return value *rc* is the full serial number to use for the next request.

For more information about the C language **Xlib** routine called by the **FXlib** binding routine **fxnextrequest**, see the **XNextRequest** routine in "Xlib Functions."

¦ Copyright IBM Corp. 1985, 1991

3.5.217 fxnoop

external **fxnoop**
integer*4 *display*

call fxnoop(*display*)

display Specifies the connection to the X Server.

fxnoop executes a **NoOperation** protocol request.

For more information about the C language **Xlib** routine called by the **FXlib** binding routine **fxnoop**, see the **XNoOp** routine in "Xlib Functions."

¦ Copyright IBM Corp. 1985, 1991

3.5.218 fxoffsetregion

external fxoffsetregion
integer*4 *r, dx, dy*

call fxoffsetregion(*r, dx, dy*)

r	Specifies the address of type **_XRegion** of the region structure.
dx, dy	Specify the *x* and *y* coordinates. These coordinates define the amount by which to move the specified region.

fxoffsetregion moves the specified region by a specified amount.

For more information about the C language **Xlib** routine called by the **FXlib** binding routine **fxoffsetregion**, see the **XOffsetRegion** routine in "Xlib Functions."

¦ Copyright IBM Corp. 1985, 1991

3.5.219 fxopendisplay

```
integer*4  fxopendisplay
external   fxopendisplay
integer*4  display
character*6 hostname

hostname = 'unix:0'

display = fxopendisplay(  hostname)
```

hostname Specifies the name of the host machine on which the display is physically attached.

fxopendisplay opens a connection to the X Server controlling the display specified.

The variable *display* contains the address of type **Display** of the returned display information.

For more information about the C language **Xlib** routine called by the **FXlib** binding routine **fxopendisplay**, see the **XOpenDisplay** routine in "Xlib Functions."

¦ Copyright IBM Corp. 1985, 1991

3.5.220 fxparsecolor

integer*4 fxparsecolor
external fxparsecolor
integer*4 *display, cmap*
character*256 *spec*
integer*4 *screendefreturn*
integer*4 *status*

status = **fxparsecolor**(*display, cmap, spec, screendefreturn*)

display	Specifies the connection to the X Server.
cmap	Specifies the colormap ID.
spec	Specifies the color-name string.
screendefreturn	Returns the address of type **XColor** that contains the values used in the colormap.

fxparsecolor parses color values.

For more information about the C language **Xlib** routine called by the **FXlib** binding routine **fxparsecolor**, see the **XParseColor** routine in "Xlib Functions."

¦ Copyright IBM Corp. 1985, 1991

3.5.221 fxparsegeometry

```
integer*4 fxparsegeometry
external  fxparsegeometry
character*256  parsestring
integer*4 xreturn, yreturn
integer*4 widthreturn, heightreturn
integer*4 changemask

changemask = fxparsegeometry( parsestring, xreturn, yreturn,
*                                          widthreturn, heightreturn)
```

parsestring Specifies the *string* to parse.

xreturn, yreturn Return the *x* offset and *y* offset determined.

widthreturn, heightreturn Return the *width* and *height* determined.

fxparsegeometry parses standard window geometry strings.

Return value *changemask* is a bit mask that indicates which geometry values were found in the specifications. It also indicates the sign of the *x* and *y* values.

For more information about the C language **Xlib** routine called by the **FXlib** binding routine **fxparsegeometry**, see the **XParseGeometry** routine in "Xlib Functions."

¦ Copyright IBM Corp. 1985, 1991

3.5.222 fxpeekevent

external fxpeekevent
integer*4 *display*
integer*4 *eventreturn*

call fxpeekevent(*display, eventreturn*)

display Specifies the connection to the X Server.

eventreturn Specifies the destination address of type **XEvent** in the client data area to copy the event structure from the event queue. The client must allocate the data area. The client must also free the data area when it is no longer needed.

fxpeekevent flushes the output buffer and peeks at the event queue.

For more information about the C language **Xlib** routine called by the **FXlib** binding routine **fxpeekevent**, see the **XPeekEvent** routine in "Xlib Functions."

¦ Copyright IBM Corp. 1985, 1991

3.5.223 fxpeekifevent

external fxpeekifevent
integer*4 *display*
integer*4 *eventreturn*
integer*4 *predicate*
character*256 *arg*

call fxpeekifevent(*display, eventreturn, predicate, arg*)

display Specifies the connection to the X Server.

eventreturn Specifies the destination address of type **XEvent** in the client data area to copy the event structure from the event queue. The client must allocate the data area. The client must also free the data area when it is no longer needed.

predicate Specifies the procedure to call to determine if the next event in the queue matches the one specified by the event argument. The procedure must be declared **external**. Only the procedure name is passed to the binding routine.

arg Specifies the address of the user-supplied argument to pass to the predicate procedure.

fxpeekifevent flushes the output buffer and checks the event queue for the event specified, but does not remove the event from the queue.

For more information about the C language **Xlib** routine called by the **FXlib** binding routine **fxpeekifevent**, see the **XPeekIfEvent** routine in "Xlib Functions."

fxpeekifevent requires a predicate procedure to pass as a parameter. The procedure determines if the event matches the one specified in the corresponding function. The predicate procedure is defined as follows:

external *functionname*
integer*4 *display*
integer*4 *event*
integer*4 *args*

call *functionname*(*display, event, args*)

functionname Specifies the user-supplied function name.

display Specifies the connection to the X Server.

event Specifies the address of type **XEvent** in the client data area.

args Specifies the arguments passed from the **fxpeekifevent** function.

The *predicate* procedure is called once for each event in the queue until it finds a match between the event in the queue and the event specified by the corresponding function. After finding a match, the predicate procedure returns **True**. If it does not find a match, it returns **False**.

¦ Copyright IBM Corp. 1985, 1991

3.5.224 fxpending

```
integer*4 fxpending
external  fxpending
integer*4 display
integer*4 rc

rc = fxpending( display)
```

display Specifies the connection to the X Server.

fxpending flushes the output buffer and returns the number of events pending.

Return value *rc* is the number of events received from the server, but not yet removed from the event queue.

For more information about the C language **Xlib** routine called by the **FXlib** binding routine **fxpending**, see the **XPending** routine in "Xlib Functions."

¦ Copyright IBM Corp. 1985, 1991

3.5.225 fxpermalloc

```
integer*4 fxpermalloc
external  fxpermalloc
integer*4 size
integer*4 rc

rc = fxpermalloc( size)
```

size Specifies the size in bytes of the area to be allocated.

fxpermalloc allocates some memory that is never freed.

Return value *rc* contains the address of the area allocated. If no area is allocated, the return value contains **NULL**.

For more information about the C language **Xlib** routine called by the **FXlib** binding routine **fxpermalloc**, see the **Xpermalloc** routine in "Xlib Functions."

¦ Copyright IBM Corp. 1985, 1991

3.5.226 fxplanesofscreen

```
integer*4  fxplanesofscreen
external   fxplanesofscreen
integer*4  screen
integer*4  rc

rc = fxplanesofscreen( screen)
```

screen Specifies the address of type **Screen** of the screen of the display.

fxplanesofscreen returns the number of planes in the screen specified.

Return value *rc* is the number of planes in the screen specified.

For more information about the C language **Xlib** routine called by the **FXlib** binding routine **fxplanesofscreen**, see the **XPlanesOfScreen** routine in "Xlib Functions."

¦ Copyright IBM Corp. 1985, 1991

3.5.227 fxpointinregion

```
integer*4 fxpointinregion
external  fxpointinregion
integer*4 r, x, y
integer*4 rc

rc = fxpointinregion( r, x, y)
```

r Specifies the address of type **_XRegion** of the region structure.

x, y Specify the *x* and *y* coordinates of the point.

fxpointinregion determines if a specified point resides in a specified region.

Return value *rc* contains a value of nonzero if point *x*, *y* is within the specified region.

For more information about the C language **Xlib** routine called by the **FXlib** binding routine **fxpointinregion**, see the **XPointInRegion** routine in "Xlib Functions."

¦ Copyright IBM Corp. 1985, 1991

3.5.228 fxpolygonregion

```
integer*4 fxpolygonregion
external  fxpolygonregion
integer*4 n
integer*4 points, fillrule
integer*4 region

region = fxpolygonregion( points, n, fillrule)
```

points Specifies an array of points.

n Specifies the number of points in the polygon.

fillrule Specifies the fill rule to set for the specified graphics context.

fxpolygonregion reads the value of any property of type **WMSIZEHINTS**.

The variable *fillrule* can be set to **EvenOddRule**.

Return value *region* is an address of type **_XRegion** that contains the specified points.

For more information about the C language **Xlib** routine called by the **FXlib** binding routine **fxpolygonregion**, see the **XPolygonRegion** routine in "Xlib Functions."

¦ Copyright IBM Corp. 1985, 1991

3.5.229 fxprotocolrevision

```
integer*4  fxprotocolrevision
external   fxprotocolrevision
integer*4  display
integer*4  rc

rc = fxprotocolrevision( display)
```

display Specifies the connection to the X Server.

fxprotocolrevision obtains the minor protocol revision number of the X Server.

Return value *rc* is the minor protocol revision number of the X Server.

For more information about the C language **Xlib** routine called by the **FXlib** binding routine **fxprotocolrevision**, see the **XProtocolRevision** routine in "Xlib Functions."

¦ Copyright IBM Corp. 1985, 1991

3.5.230 fxprotocolversion

integer*4 fxprotocolversion
external   fxprotocolversion
integer*4 *display*
integer*4 *rc*

rc = **fxprotocolversion**(*display*)

display Specifies the connection to the X Server.

fxprotocolversion obtains the major version number (11) of the X protocol associated with the display connected.

Return value *rc* is the major version number.

For more information about the C language **Xlib** routine called by the **FXlib** binding routine **fxprotocolversion**, see the **XProtocolVersion** routine in "Xlib Functions."

¦ Copyright IBM Corp. 1985, 1991

3.5.231 fxputbackevent

external fxputbackevent
integer*4 *display*
integer*4 *event*

call fxputbackevent(*display, event*)

display Specifies the connection to the X Server.

event Specifies the destination address of type **XEvent** in the client data area to copy the event structure into the event queue. The client must allocate the data area. The client must also free the data area when it is no longer needed.

fxputbackevent pushs an event back to the top of the event queue.

For more information about the C language **Xlib** routine called by the **FXlib** binding routine **fxputbackevent**, see the **XPutBackEvent** routine in "Xlib Functions."

¦ Copyright IBM Corp. 1985, 1991

3.5.232 fxputimage

```
external  fxputimage
integer*4 display
integer*4 drawable, gc image
integer*4 srcx, srcy, dstx, dsty
integer*4 width, height

call fxputimage( display, drawable, gc image, srcx, srcy, dstx, dsty, width,
*                    height)
```

display Specifies the connection to the X Server.

drawable Specifies the drawable.

gc Specifies the address of type **GC** that has the graphics context.

image Specifies the address of type XImage that has the image to be combined with the rectangle.

srcx Specifies the offset in X from the left edge of the image defined by the **XImage** data structure.

srcy Specifies the offset in X from the top edge of the image defined by the **XImage** data structure.

dstx, dsty Specify the *x* and *y* coordinates of the subimage relative to the origin of the drawable where the image is drawn.

width, height Specify the *width* and *height* of the subimage. These arguments define the dimensions of the rectangle.

fxputimage combines an image in memory with a rectangle of a drawable on the display.

This routine can generate the errors **BadDrawable**, **BadValue**, **BadMatch**, and **BadGC**.

For more information about the C language **Xlib** routine called by the **FXlib** binding routine **fxputimage**, see the **XPutImage** routine in "Xlib Functions."

¦ Copyright IBM Corp. 1985, 1991

3.5.233 fxputpixel

```
integer*4 fxputpixel
external  fxputpixel
integer*4 ximage, x, y, pixel
integer*4 status

status = fxputpixel( ximage, x, y, pixel)
```

ximage Specifies the address of type **XImage** of the image structure.

x, y Specify the *x* and *y* coordinates.

pixel Specifies the new pixel value.

fxputpixel sets a pixel value in an image.

Return value *status* indicates the success of the function.

For more information about the C language **Xlib** routine called by the **FXlib** binding routine **fxputpixel**, see the **XPutPixel** routine in "Xlib Functions."

¦ Copyright IBM Corp. 1985, 1991

3.5.234 fxputstring

external fxputstring
integer*4 *startaddress*
integer*4 *offset*
character*256 *string*

call fxputstring(*startaddress, offset, string*)

startaddress Specifies an address in memory used to determine an intermediate address which will contain the address of the string.

offset Specifies the offset in bytes from *startaddress* to the intermediate address.

string Specifies the string to store.

fxputstring puts a string into memory. The address of where the string is stored is determined by the routine. The routine then stores the string address at an address specified by *startaddress* plus *offset*.

¦ Copyright IBM Corp. 1985, 1991

3.5.235 fxputvalue

external fxputvalue

integer*4 *startaddress*
integer*4 *offset*
integer*4 *length*
integer*4 *value*

call fxputvalue(*startaddress*, *offset*, *length*, *value*)

startaddress	Specifies the starting address in memory used to determine the address that stores the value.
offset	Specifies the offset in bytes from *startaddress* to the address that stores the value.
length	Specifies the length of *value* in bytes (1, 2, or 4).
value	Specifies the value to store.

fxputvalue puts a 1-, 2-, or 4-byte value into memory. The *startaddress* plus the *offset* provides an intermediate address that points to the location in memory to put the value.

¦ Copyright IBM Corp. 1985, 1991

3.5.236 fxqlength

```
integer*4  fxqlength
external   fxqlength
integer*4  display
integer*4  rc

rc = fxqlength( display)
```

display Specifies the connection to the X Server.

fxqlength obtains the length of the event queue for the display connected.

Return value *rc* is the length of the event queue.

For more information about the C language **Xlib** routine called by the **FXlib** binding routine **fxqlength**, see the **XQLength** routine in "Xlib Functions."

¦ Copyright IBM Corp. 1985, 1991

3.5.237 fxquerybestcursor

```
integer*4 fxquerybestcursor
external  fxquerybestcursor
integer*4 display
integer*4 drawable, width, height
integer*4 widthreturn, heightreturn
integer*4 status

status = fxquerybestcursor( display, drawable, width, height,
*                                   widthreturn, heightreturn)
```

display	Specifies the connection to the X Server.
drawable	Specifies the drawable. The drawable indicates the desired screen.
width	Specifies the desired cursor width.
widthreturn	Returns the hardware-supported width that is closest to *width*.
height	Specifies the desired cursor height.
heightreturn	Returns the hardware-supported height that is closest to *height*.

fxquerybestcursor determines useful cursor sizes.

This routine can generate the error **BadDrawable**.

For more information about the C language **Xlib** routine called by the **FXlib** binding routine **fxquerybestcursor**, see the **XQueryBestCursor** routine in "Xlib Functions."

¦ Copyright IBM Corp. 1985, 1991

3.5.238 fxquerybestsize

```
integer*4 fxquerybestsize
external  fxquerybestsize
integer*4 display
integer*4 class
integer*4 whichscreen
integer*4 width, height
integer*4 widthreturn, heightreturn
integer*4 status

status = fxquerybestsize( display, class, whichscreen, width,
*                                 height, widthreturn, heightreturn)
```

display Specifies the connection to the X Server.

class Specifies the class of object being sized.

whichscreen Specifies any drawable on a screen.

width Specifies the desired width of the object.

widthreturn Returns the closest width of the object supported by the display hardware.

height Specifies the desired height of the object.

heightreturn Returns the closest height of the object supported by the display hardware.

For the specified object class, **fxquerybestsize** returns the hardware-supported size that is closest to the requested size.

The variable *class* can be set to **TileShape**, **CursorShape**, or **StippleShape**.

This routine can generate the errors **BadDrawable**, **BadMatch**, and **BadValue**.

For more information about the C language **Xlib** routine called by the **FXlib** binding routine **fxquerybestsize**, see the **XQueryBestSize** routine in "Xlib Functions."

¦ Copyright IBM Corp. 1985, 1991

3.5.239 fxquerybeststipple

```
integer*4 fxquerybeststipple
external  fxquerybeststipple
integer*4 display
integer*4 whichscreen
integer*4 width, height
integer*4 widthreturn, heightreturn
integer*4 status

status = fxquerybeststipple( display, whichscreen, width,
*                                    height, widthreturn, heightreturn)
```

display Specifies the connection to the X Server.

whichscreen Specifies any drawable on a screen.

width Specifies the desired stipple width.

widthreturn Returns the hardware-supported stipple width that is closest to *width*.

height Specifies the desired stipple height.

heightreturn Returns the hardware-supported stipple height that is closest to *height*.

fxquerybeststipple obtains the best stipple shape.

This routine can generate the error **BadDrawable**.

For more information about the C language **Xlib** routine called by the **FXlib** binding routine **fxquerybeststipple**, see the **XQueryBestStipple** routine in "Xlib Functions."

¦ Copyright IBM Corp. 1985, 1991

3.5.240 fxquerybesttile

```
integer*4 fxquerybesttile
external  fxquerybesttile
integer*4 display
integer*4 whichscreen
integer*4 width, height
integer*4 widthreturn, heightreturn
integer*4 status

status = fxquerybesttile( display, whichscreen, width, height,
*                                 widthreturn, heightreturn)
```

display Specifies the connection to the X Server.

whichscreen Specifies any drawable on a screen.

width Specifies the desired tile width.

widthreturn Returns the hardware-supported tile width that is closest to *width*.

height Specifies the desired tile height.

heightreturn Returns the hardware-supported tile height that is closest to *height*.

fxquerybesttile obtains the best fill tile shape.

This routine can generate the error **BadDrawable**.

For more information about the C language **Xlib** routine called by the **FXlib** binding routine **fxquerybesttile**, see the **XQueryBestTile** routine in "Xlib Functions."

¦ Copyright IBM Corp. 1985, 1991

3.5.241 fxquerycolor

external fxquerycolor
integer*4 *display*
integer*4 *cmap*
integer*4 *defreturn*

call fxquerycolor(*display, cmap, defreturn*)

display Specifies the connection to the X Server.

cmap Specifies the colormap ID.

defreturn Returns an address of type **XColor** that has the RGB values for the pixel specified in the structure.

fxquerycolor obtains the color values for a single specified pixel value.

This routine can generate the errors **BadValue** and **BadColor**.

For more information about the C language **Xlib** routine called by the **FXlib** binding routine **fxquerycolor**, see the **XQueryColor** routine in "Xlib Functions."

¦ Copyright IBM Corp. 1985, 1991

3.5.242 fxquerycolors

external fxquerycolors
integer*4 *display*
integer*4 *cmap*
integer*4 *defsreturn, ncolors*

call fxquerycolors(*display, cmap, defsreturn, ncolors*)

display Specifies the connection to the X Server.

cmap Specifies the colormap ID.

defreturn Returns an address of type **XColor** that has an array of color definition data structures.

ncolors Specifies the number of type **XColor** data structures in the color definition array.

fxquerycolors obtains the color values for an array of pixels stored in color data structures.

This routine can generate errors **BadValue** and **BadColor**.

For more information about the C language **Xlib** routine called by the **FXlib** binding routine **fxquerycolors**, see the **XQueryColors** routine in "Xlib Functions."

¦ Copyright IBM Corp. 1985, 1991

3.5.243 fxqueryfont

```
integer*4 fxqueryfont
external  fxqueryfont
integer*4 display
integer*4 fontid
integer*4 font

font = fxqueryfont( display, fontid)
```

display Specifies the connection to the X Server.

fontid Specifies the ID of the font or the graphics context of the font to be defined.

fxqueryfont obtains information about a loaded font.

Return value *font* is an address of type **XFontStruct** that has the font structure information.

For more information about the C language **Xlib** routine called by the **FXlib** binding routine **fxqueryfont**, see the **XQueryFont** routine in "Xlib Functions."

¦ Copyright IBM Corp. 1985, 1991

3.5.244 fxquerykeymap

external fxquerykeymap
integer*4 *display*
integer*4 *keysreturn*

call fxquerykeymap(*display, keysreturn*)

display Specifies the connection to the X Server.

keysreturn Returns an array of 32-bit words that identifies which keys are pressed. Each bit represents one keyboard key.

fxquerykeymap gets a bit vector for the keyboard.

For more information about the C language **Xlib** routine called by the **FXlib** binding routine **fxquerykeymap**, see the **XQueryKeymap** routine in "Xlib Functions."

¦ Copyright IBM Corp. 1985, 1991

3.5.245 fxquerypointer

```
integer*4 fxquerypointer
external  fxquerypointer
integer*4 display
integer*4 window
integer*4 rootreturn, childreturn
integer*4 rootxreturn, rootyreturn
integer*4 winxreturn, winyreturn
integer*4 maskreturn
integer*4 rc

rc = fxquerypointer( display, window, rootreturn, childreturn,
*                    rootxreturn, rootyreturn, winxreturn,
*                    winyreturn, maskreturn)
```

display	Specifies the connection to the X Server.
window	Specifies the window ID.
rootreturn	Returns the root window ID for the window specified which identifies the root window the pointer is currently on.
childreturn	Returns the child window ID that the pointer is located in, if any.
rootxreturn, rootyreturn	Return the pointer coordinates relative to the origin of the root window.
winxreturn, winyreturn	Return the pointer coordinates relative to the window specified.
maskreturn	Returns the current state of the modifier keys and pointer buttons.

fxquerypointer obtains the root window the pointer is currently on and the pointer coordinates relative to the origin of the root window.

Return value *rc* contains a value describing the relationship of the pointers to the screen and window.

If **fxquerypointer** returns **FALSE**, the pointer is not on the same screen as the window associated with the *window* argument. It returns the constant **None** to *childreturn* and zero to *winxreturn* and *winyreturn*.

If **fxquerypointer** returns **TRUE**, the pointer coordinates to *winxreturn* and *winyreturn* are relative to the origin of the window identified by the *window* argument. It returns the ID of the child containing the pointer, if any.

fxquerypointer can generate the error **BadWindow**.

For more information about the C language **Xlib** routine called by the **FXlib** binding routine **fxquerypointer**, see the **XQueryPointer** routine in "Xlib Functions."

¦ Copyright IBM Corp. 1985, 1991

3.5.246 fxquerytextextents

external fxquerytextextents
integer*4 *display*
integer*4 *fontID*
character*256 *string*
integer*4 *ncharacters, directionreturn, ascentreturn*
integer*4 *descentreturn, overallreturn*

call fxquerytextextents(*display, fontID, string, ncharac ters,*
** directionreturn, ascentreturn,*
** descentreturn, overallreturn*)

display Specifies the connection to the X Server.

fontID Specifies the font ID or the graphics context that contains the font.

string Specifies the character string of 8-bit characters.

ncharacters Specifies the number of characters in the character string.

directionreturn Returns the value of the direction hint member **FontLeftToRight** or **FontRightToLeft**.

ascentreturn Returns the ascent member.

descentreturn Returns the descent member.

overallreturn Returns the address of type **XCharStruct** that has the overall size in the specified data structure.

fxquerytextextents queries the server for the sizes of an 8-bit character string.

For more information about the C language **Xlib** routine called by the **FXlib** binding routine **fxquerytextextents**, see the **XQueryTextExtents** routine in "Xlib Functions."

external fxquerytextextents16
integer*4 *display*
integer*4 *fontID*
integer*4 *string*
integer*4 *ncharacters,directionreturn,ascentreturn*
integer*4 *descentreturn,overallreturn*

call fxquerytextextents16(*display, fontID, string,*
** ncharacters directionreturn,ascentreturn,*
** descentreturn,overallreturn*)

display Specifies the connection to the X Server.

fontID Specifies the font ID or the graphics context that contains the font.

string Specifies the address of type **XChar2b** that has a character string of 16-bit characters.

ncharacters Specifies the number of characters in the character

¦ Copyright IBM Corp. 1985, 1991

string.

directionreturn Returns the value of the direction hint member **FontLeftToRight** or **FontRightToLeft**.

ascentreturn Returns the ascent member.

descentreturn Returns the descent member.

overallreturn Returns the address of type **XCharStruct** that has the overall size in the specified data structure.

fxquerytextextents16 queries the server for the sizes of a 16-bit character string.

For more information about the C language **Xlib** routine called by the **FXlib** binding routine **fxquerytextextents16**, see the **XQueryTextExtents16** routine in "Xlib Functions."

¦ Copyright IBM Corp. 1985, 1991

3.5.247 fxquerytree

```
integer*4 fxquerytree
external  fxquerytree
integer*4 display
integer*4 window
integer*4 rootreturn, parentreturn
integer*4 childrenreturn
integer*4 nchildrenreturn
integer*4 status

status = fxquerytree( display, window, rootreturn, parentreturn,
*                             childrenreturn, nchildrenreturn)
```

display Specifies the connection to the X Server.

windowindow Specifies the window ID.

rootreturn Returns the root window ID for the specified window.

parentreturn Returns the parent window ID for the specified window.

childrenreturn Returns the starting address of a list of children for the specified window.

nchildrenreturn Returns the number of children for the specified window.

fxquerytree obtains a list of children, the parent, and the number of children for a specified window.

Return value *status* contains the status of the query. If the query fails, it returns a zero. If the query succeeds, it returns a non-zero.

For more information about the C language **Xlib** routine called by the **FXlib** binding routine **fxquerytree**, see the **XQueryTree** routine in "Xlib Functions."

¦ Copyright IBM Corp. 1985, 1991

3.5.248 fxraisewindow

external fxraisewindow
integer*4 *display*
integer*4 *window*

call fxraisewindow(*display, window*)

display Specifies the connection to the X Server.

window Specifies the window ID.

fxraisewindow raises a window so that it is not hidden by a sibling window.

For more information about the C language **Xlib** routine called by the **FXlib** binding routine **fxraisewindow**, see the **XRaiseWindow** routine in "Xlib Functions."

¦ Copyright IBM Corp. 1985, 1991

3.5.249 fxreadbitmapfile

```
integer*4 fxreadbitmapfile
external  fxreadbitmapfile
integer*4 display, drawable
character*256  filename
integer*4 widthreturn, heightreturn, bitmapreturn
integer*4 xhotreturn, yhotreturn
integer*4 status

status = fxreadbitmapfile( display, drawable, filename, widthreturn,
*                          heightreturn, bitmapreturn, xhotreturn,
*                          yhotreturn)
```

display	Specifies the connection to the X Server.
drawable	Specifies the drawable.
filename	Specifies the file name to use. The format of the file name is specific to the operating system.
widthreturn, heightreturn	Return the *width* and *height* values of the bitmap file.
bitmapreturn	Returns the bitmap ID that is created.
xhotreturn, yhotreturn	Return the hot spot coordinates.

fxreadbitmapfile reads a bitmap from disk.

For more information about the C language **Xlib** routine called by the **FXlib** binding routine **fxreadbitmapfile**, see the **XReadBitmapFile** routine in "Xlib Functions."

¦ Copyright IBM Corp. 1985, 1991

3.5.250 fxrebindcode

external fxrebindcode
integer*4 *display*
integer*4 *keycode*
integer*4 *shiftbits*
integer*4 *string*
integer*4 *nbytes*

call fxrebindcode (*display, keycode, shiftbits, string, nbytes*)

display Specifies the connection to the server.

keycode Specifies the keycode to temporarily change.

shiftbits Specifies the shift bits.

string Specifies the address of the string to assign.

nbytes Specifies the number of bytes in the string.

fxrebindcode changes the binding of the keyboard on a non-permanent basis.

For more information about the C language **Xlib** routine called by the **FXlib** binding routine **fxrebindcode**, see the **XRebindCode** routine in "Xlib Functions" in topic 2.4.

¦ Copyright IBM Corp. 1985, 1991

3.5.251 fxrebindkeysym

```
external  fxrebindkeysym
integer*4 display
integer*4 keysym, list, modcount
character*256 string
integer*4 bytesstring

call fxrebindkeysym( display, keysym, list, modcount, string,
*                            bytesstring)
```

display Specifies the connection to the X Server.

keysym Specifies the *keysym* to be rebound.

list Specifies an address of an array of *keysyms* that are being used as modifiers.

modcount Specifies the number of modifiers in the modifier list.

string Specifies an address of the *string* to be returned by **fxlookupstring**.

bytesstring Specifies the length of the string.

fxrebindkeysym rebinds the meaning of a keysym for a client.

For more information about the C language **Xlib** routine called by the **FXlib** binding routine **fxrebindkeysym**, see the **XRebindKeysym** routine in "Xlib Functions."

¦ Copyright IBM Corp. 1985, 1991

3.5.252 fxrecolorcursor

```
external  fxrecolorcursor
integer*4 display
integer*4 cursor
integer*4 foregroundcolor, backgroundcolor

call fxrecolorcursor( display, cursor,
*                         foregroundcolor, backgroundcolor)
```

display Specifies the connection to the X Server.

cursor Specifies the cursor ID.

foregroundcolor Specifies the address of type **XColor** that has the RGB values for the foreground of the source.

backgroundcolor Specifies the address of type **XColor** that has the RGB values for the background of the source.

fxrecolorcursor changes the color of the specified cursor.

This routine can generate the error **BadCursor**.

For more information about the C language **Xlib** routine called by the **FXlib** binding routine **fxrecolorcursor**, see the **XRecolorCursor** routine in "Xlib Functions."

¦ Copyright IBM Corp. 1985, 1991

3.5.253 fxrectinregion

```
integer*4 fxrectinregion
external  fxrectinregion
integer*4 r, x, y, width, height
integer*4 rc

rc = fxrectinregion( r, x, y, width, height)
```

r Specifies the address of type **_XRegion** of the region structure.

x, y Specify the *x* and *y* coordinates of the point.

width, height Specify the *width* and *height* of the rectangle to determine if the point resides within this region.

fxrectinregion determines if a specified rectangle resides in a specified region.

Return value *rc* can be one of the following:

RectangleIn, if the rectangle is entirely in the specified region.
RectangleOut, if the rectangle is entirely out of the specified region.
RectanglePart, if the rectangle is is partially in the specified region.

For more information about the C language **Xlib** routine called by the **FXlib** binding routine **fxrectinregion**, see the **XRectInRegion** routine in "Xlib Functions."

¦ Copyright IBM Corp. 1985, 1991

3.5.254 fxrefreshkeyboardmapping

```
external  fxrefreshkeyboardmapping
integer*4 eventmap

call fxrefreshkeyboardmapping( eventmap)
```

eventmap Specifies the address of type **XMappingEvent** that contains the mapping event to be used.

fxrefreshkeyboardmapping refreshes the stored modifier and keymap information.

For more information about the C language **Xlib** routine called by the **FXlib** binding routine **fxrefreshkeyboardmapping**, see the **XRefreshKeyboardMapping** routine in "Xlib Functions."

¦ Copyright IBM Corp. 1985, 1991

3.5.255 fxremovefromsaveset

```
external fxremovefromsaveset
integer*4 display
integer*4 windowremove

call fxremovefromsaveset( display, windowremove)
```

display Specifies the connection to the X Server.

windowremove Specifies the window ID of the window whose children are to be removed from the client save-set.

fxremovefromsaveset removes a window from the client save-set.

This routine can generate the errors **BadWindow** and **BadMatch**.

For more information about the C language **Xlib** routine called by the **FXlib** binding routine **fxremovefromsaveset**, see the **XRemoveFromSaveSet** routine in "Xlib Functions."

¦ Copyright IBM Corp. 1985, 1991

3.5.256 fxremovehost

external fxremovehost
integer*4 *display, host*

call fxremovehost(*display, host*)

display Specifies the connection to the X Server.

host Specifies the address of type **XHostAddress** of the the host machine to be removed from the network access control list.

fxremovehost removes a single host.

This routine can generate the errors **BadAlloc** and **BadValue**.

For more information about the C language **Xlib** routine called by the **FXlib** binding routine **fxremovehost**, see the **XRemoveHost** routine in "Xlib Functions."

¦ Copyright IBM Corp. 1985, 1991

3.5.257 fxremovehosts

external fxremovehosts
integer*4 *display, hosts, numhosts*

call fxremovehosts(*display, hosts, numhosts*)

display Specifies the connection to the X Server.

hosts Specifies the starting address of type **XHostAddress** for a list of hosts that are to be removed from the network access control list. The client must allocate the host list and free the host list when it is no longer needed.

numhosts Specifies the number of hosts.

fxremovehosts removes multiple hosts at one time.

This routine can generate the errors **BadAlloc** and **BadValue** errors.

For more information about the C language **Xlib** routine called by the **FXlib** binding routine **fxremovehosts**, see the **XRemoveHosts** routine in "Xlib Functions."

¦ Copyright IBM Corp. 1985, 1991

3.5.258 fxreparentwindow

external fxreparentwindow
integer*4 *display*
integer*4 *window*
integer*4 *parent*
integer*4 *x, y*

call fxreparentwindow(*display, window, parent, x, y*)

display Specifies the connection to the X Server.

window Specifies the window ID.

parent Specifies the parent window ID.

x, y Specify the *x* and *y* coordinates of the position in the new parent window of the specified window.

fxreparentwindow changes the parent of a window within a single screen. Windows cannot be moved between screens.

This routine can generate the error **BadWindow**.

For more information about the C language **Xlib** routine called by the **FXlib** binding routine **fxreparentwindow**, see the **XReparentWindow** routine in "Xlib Functions."

¦ Copyright IBM Corp. 1985, 1991

3.5.259 fxresetscreensaver

external fxresetscreensaver
integer*4 *display*

call fxresetscreensaver(*display*)

display Specifies the connection to the X Server.

fxresetscreensaver resets the screen saver.

For more information about the C language **Xlib** routine called by the **FXlib** binding routine **fxresetscreensaver**, see the **XResetScreenSaver** routine in "Xlib Functions."

¦ Copyright IBM Corp. 1985, 1991

3.5.260 fxresizewindow

external fxresizewindow
integer*4 *display*
integer*4 *window*
integer*4 *width*
integer*4 *height*

call fxresizewindow (*display*, *window*, *width*, *height*)

display	Specifies the connection to the X Server.
window	Specifies the window ID.
width, height	Specify the *width* and *height* dimensions of the window after the call is completed.

fxresizewindow changes the size of a window without changing the upper-left coordinate.

For more information about the C language **Xlib** routine called by the **FXlib** binding routine **fxresizewindow**, see the **XResizeWindow** routine in "Xlib Functions."

¦ Copyright IBM Corp. 1985, 1991

3.5.261 fxresourcemanagerstring

```
integer*4 fxresourcemanagerstring
external  fxresourcemanagerstring
integer*4 display
integer*4 string

string = fxresourcemanagerstring( display)
```

display Specifies the connection to the X Server.

fxresourcemanagerstring returns the **RESOURCE_MANAGER** property from the screen of the root window of screen zero. This is the value returned by **fxopendisplay**.

For more information about the C language **Xlib** routine called by the **FXlib** binding routine **fxresourcemanagerstring**, see the **XResourceManagerString** routine in "Xlib Functions."

¦ Copyright IBM Corp. 1985, 1991

3.5.262 fxrestackwindows

external fxrestackwindows
integer*4 *display*
integer*4 *window*
integer*4 *nwindows*

call fxrestackwindows(*display, window, nwindows*)

display Specifies the connection to the X Server.

window Specifies the window ID.

nwindows Specifies the number of windows to restack.

fxrestackwindows restacks a set of windows from top to bottom.

For more information about the C language **Xlib** routine called by the **FXlib** binding routine **fxrestackwindows**, see the **XRestackWindows** routine in "Xlib Functions."

¦ Copyright IBM Corp. 1985, 1991

3.5.263 fxrmgetfiledatabase

```
integer*4 fxrmgetfiledatabase
external  fxrmgetfiledatabase
character*256 dbfilename
integer*4 rdb

rdb = fxrmgetfiledatabase( dbfilename)
```

dbfilename Specifies the resource database filename.

fxrmgetfiledatabase retrieves a database from nonvolatile storage.

Return value *rdb* is the address of type **XrmResourceDatabase** that contains the resource database structure.

For more information about the C language **Xlib** routine called by the **FXlib** binding routine **fxrmgetfiledatabase**, see the **XrmGetFileDatabase** routine in "Xlib Functions."

¦ Copyright IBM Corp. 1985, 1991

3.5.264 fxrmgetresource

```
integer*4 fxrmgetresource
external  fxrmgetresource
integer*4 db
character*256 strname
character*256 strclass
integer*4 strtypereturn
integer*4 strvaluereturn
integer*4 bool

bool = fxrmgetresource( db, strname, strclass, strtypereturn,
*                                  strvaluereturn)
```

db Specifies the address of type **XrmDatabase** of the descriptor of the resource database to be used.

strname Specifies as a string the fully qualified name of the value being retrieved.

strclass Specifies as a string the fully qualified class of the value being retrieved.

strtypereturn Returns as a string the address of the representation type of the destination.

strvaluereturn Returns the value in the database.

fxrmgetresource retrieves a resource from a resource database.

For more information about the C language **Xlib** routine called by the **FXlib** binding routine **fxrmgetresource**, see the **XrmGetResource** routine in "Xlib Functions."

¦ Copyright IBM Corp. 1985, 1991

3.5.265 fxrmgetstringdatabase

```
integer*4 fxrmgetstringdatabase
external  fxrmgetstringdatabase
character*256 data
integer*4 rdb

rdb = fxrmgetstringdatabase( data)
```

data Specifies the database contents using a string.

fxrmgetstringdatabase creates a database from a string.

Return value *rdb* is the address of type **XrmResourceDatabase** that contains the resource database structure.

For more information about the C language **Xlib** routine called by the **FXlib** binding routine **fxrmgetstringdatabase**, see the **XrmGetStringDatabase** routine in "Xlib Functions."

¦ Copyright IBM Corp. 1985, 1991

3.5.266 fxrminitialize

external fxrminitialize

call fxrminitialize()

fxrminitialize initializes the resource manager.

For more information about the C language **Xlib** routine called by the **FXlib** binding routine **fxrminitialize**, see the **XrmInitialize** routine in "Xlib Functions."

¦ Copyright IBM Corp. 1985, 1991

3.5.267 fxrmmergedatabases

external fxrmmergedatabases
integer*4 *sourcedb*
integer*4 *targetdb*

call fxrmmergedatabases(*sourcedb, targetdb*)

sourcedb Specifies the address of type **XrmResourceDatabase** that contains the descriptor of the resource database to be merged into the target database.

targetdb Specifies the address of type **XrmResourceDatabase** that contains the descriptor of the resource database into which the source database is merged.

fxrmmergedatabases merges the contents of one database into another.

For more information about the C language **Xlib** routine called by the **FXlib** binding routine **fxrmmergedatabases**, see the **XrmMergeDatabases** routine in "Xlib Functions."

¦ Copyright IBM Corp. 1985, 1991

3.5.268 fxrmparsecommand

```
external  fxrmparsecommand
integer*4 db, table, tablecount
character*256  name
integer*4 argcreturn, argvreturn

call fxrmparsecommand( db, table, tablecount, name, argcreturn,
*                                 argvreturn)
```

db Specifies the address of the resource database. If it contains **NULL**, a new resource database is created. The address of the new resource data is returned in *db*.

table Specifies the address of **XrmOptionDescList** of the table of command line arguments to be parsed.

tablecount Specifies the number of entries in the table.

name Specifies the application name.

argcreturn Before the call, *argcreturn* contains the number of arguments. After the call, *arcreturn* returns the number of remaining arguments.

argvreturn Before the call, *argvreturn* contains an address of the command line arguments. After the call, *argvreturn* contains unmatched arguments only.

fxrmparsecommand loads a resource database from a C language command line.

For more information about the C language **Xlib** routine called by the **FXlib** binding routine **fxrmparsecommand**, see the **XrmParseCommand** routine in "Xlib Functions."

¦ Copyright IBM Corp. 1985, 1991

3.5.269 fxrmputfiledatabase

external fxrmputfiledatabase
integer*4 *db*
character*256 *storeddb*

call fxrmputfiledatabase(*db, storeddb*)

db Specifies the resource database to be used.

storeddb Specifies the filename for the stored database.

fxrmputfiledatabase stores a copy of the application's current database in nonvolatile storage.

For more information about the C language **Xlib** routine called by the **FXlib** binding routine **fxrmputfiledatabase**, see the **XrmPutFileDatabase** routine in "Xlib Functions."

¦ Copyright IBM Corp. 1985, 1991

3.5.270 fxrmputlineresource

external fxrmputlineresource
integer*4 *db*
character*256 *line*

call fxrmputlineresource(*db, line*)

db Specifies the address of the resource database. If it contains **NULL**, a new resource database is created. The address to the database is returned in *db*.

line Specifies the resource value pair as a single string. A single colon (:) separates the name from the value.

fxrmputlineresource adds a single resource entry that is specified as a string that contains both a name and a value.

For more information about the C language **Xlib** routine called by the **FXlib** binding routine **fxrmputlineresource**, see the **XrmPutLineResource** routine in "Xlib Functions."

¦ Copyright IBM Corp. 1985, 1991

3.5.271 fxrmputresource

external fxrmputresource
integer*4 *db*
character*256 *spec*
character*256 *type*
integer*4 *value*

call fxrmputresource(*db, spec, type, value*)

db Specifies the address of the resource database. If it contains **NULL**, a new resource database is created and the address is returned in *db*.

spec Specifies a partial specification of the resource.

type Specifies the *type* of the resource.

value Specifies the *value* of the resource.

fxrmputresource stores resources into the database.

For more information about the C language **Xlib** routine called by the **FXlib** binding routine **fxrmputresource**, see the **XrmPutResource** routine in "Xlib Functions."

¦ Copyright IBM Corp. 1985, 1991

3.5.272 fxrmputstringresource

external fxrmputstringresource
integer*4 *db*
character*256 *resource*
character*256 *value*

call fxrmputstringresource(*db, resource, value*)

db Specifies the address of the resource database. If it contains **NULL**, a new resource database is created. The address to the database is returned in *db*.

resource Specifies the resource as a string.

value Specifies as a string the *value* of the resource.

fxrmputstringresource adds a resource that is specified as a string.

For more information about the C language **Xlib** routine called by the **FXlib** binding routine **fxrmputstringresource**, see the **XrmPutStringResource** routine in "Xlib Functions."

¦ Copyright IBM Corp. 1985, 1991

3.5.273 fxrmqgetresource

```
integer*4 fxrmqgetresource
external  fxrmqgetresource
integer*4 db
integer*4 quarkname
integer*4 quarkclass
integer*4 quarktypereturn
integer*4 valuereturn
integer*4 bool

bool = fxrmqgetresource( db, quarkname, quarkclass,
*                                quarktypereturn, valuereturn)
```

db Specifies the address of **XrmDatabase** of the descriptor of the resource database to be used.

quarkname Specifies the fully qualified name of the value being retrieved as a quark.

quarkclass Specifies the fully qualified class of the value being retrieved as a quark.

quarktypereturn Returns the address of the representation type of the destination as a quark.

valuereturn Returns the value in the database.

fxrmqgetresource retrieves a resource from a resource database.

For more information about the C language **Xlib** routine called by the **FXlib** binding routine **fxrmqgetresource**, see the **XrmQGetResource** routine in "Xlib Functions."

¦ Copyright IBM Corp. 1985, 1991

3.5.274 fxrmqgetsearchlist

```
integer*4 fxrmqgetsearchlist
external  fxrmqgetsearchlist
integer*4 db
integer*4 names
integer*4 classes
integer*4 listreturn
integer*4 listlength
integer*4 bool

bool = fxrmqgetsearchlist( db, names, classes, listreturn,
*                                  listlength)
```

db Specifies an address of type **XrmDatabase** of the database to be used.

names Specifies a starting address of type **XrmNameList** of a list of resource names.

classes Specifies a starting address of type **XrmClassList** of a list of resource classes.

listreturn Returns a search list for further use. The client must allocate the resource list and must free it when it is no longer needed.

listlength Specifies the number of entries, not the byte size, allocated for *listreturn*.

fxrmqgetsearchlist returns a list of database levels.

For more information about the C language **Xlib** routine called by the **FXlib** binding routine **fxrmqgetsearchlist**, see the **XrmQGetSearchList** routine in "Xlib Functions."

¦ Copyright IBM Corp. 1985, 1991

3.5.275 fxrmqgetsearchresource

```
integer*4 fxrmqgetsearchresource
external  fxrmqgetsearchresource
integer*4 list, name
integer*4 class, typereturn, valuereturn
integer*4 bool

bool = fxrmqgetsearchresource( list, name, class, typereturn,
*                                          valuereturn)
```

list Specifies the starting address of the search list returned by **fxrmqgetsearchlist**.

name Specifies the resource name.

class Specifies the resource class.

typereturn Returns the data representation type.

valuereturn Returns the value in the database.

fxrmqgetsearchresource searches the database levels for a given resource.

Return variable *bool* returns **TRUE** if the resource is found.

For more information about the C language **Xlib** routine called by the **FXlib** binding routine **fxrmqgetsearchresource**, see the **XrmQGetSearchResource** routine in "Xlib Functions."

¦ Copyright IBM Corp. 1985, 1991

3.5.276 fxrmqputresource

external fxrmqputresource
integer*4 *db*
integer*4 *bindings*
integer*4 *quarks*
integer*4 *type*
integer*4 *value*

call fxrmqputresource(*db, bindings, quarks, type, value*)

db Specifies the address of the resource database. If it contains **NULL**, a new resource database is created and the address to it is returned in *db*.

bindings Specifies a list of bindings.

quarks Specifies the partial name or class list of the resource to be stored.

type Specifies the *type* of the resource.

value Specifies the *value* of the resource.

fxrmqputresource stores resources into the database.

For more information about the C language **Xlib** routine called by the **FXlib** binding routine **fxrmqputresource**, see the **XrmQPutResource** routine in "Xlib Functions."

¦ Copyright IBM Corp. 1985, 1991

3.5.277 fxrmqputstringresource

external fxrmqputstringresource
integer*4 *db*
integer*4 *bindings*
integer*4 *quarks*
character*256 *value*

call fxrmqputstringresource(*db, bindings, quark, value*)

db Specifies the address of the resource database. If it contains **NULL**, a new resource database is created. The address to the database is returned in *db*. If the resource database is **NULL**, a new database is created.

bindings Specifies a list of bindings.

quarks Specifies the partial name or class list of the resource to be stored.

value Specifies the value of the resource. The value is specified as a string.

fxrmqputstringresource adds a string resource using quarks as a specification.

For more information about the C language **Xlib** routine called by the **FXlib** binding routine **fxrmqputstringresource**, see the **XrmQPutStringResource** routine in "Xlib Functions."

¦ Copyright IBM Corp. 1985, 1991

3.5.278 fxrmquarktostring

```
character*256 fxrmquarktostring
external  fxrmquarktostring
integer*4 quark
character*256 qtostring

qtostring = fxrmquarktostring( quark)
```

quark Specifies the *quark* for the equivalent string desired.

fxrmquarktostring converts a quark to a string.

Return value *qtostring* contains the string for the specified *quark*.

For more information about the C language **Xlib** routine called by the **FXlib** binding routine **fxrmquarktostring** see the **XrmQuarkToString** routine in "Xlib Functions."

¦ Copyright IBM Corp. 1985, 1991

3.5.279 fxrmstringtobindingquarklist

```
external  fxrmstringtobindingquarklist
character*256 string
integer*4 bindingsreturn
integer*4 quarksreturn

call fxrmstringtobindingquarklist( string, bindingsreturn,
*                                   quarksreturn)
```

string	Specifies the *string* for a quark to be allocated.
bindingsreturn	Returns the binding list. The client must allocate the binding list and must free the binding list when it is no longer needed.
quarksreturn	Returns the list of quarks. The client must allocate the quark list and must free the quark list when it is no longer needed.

fxrmstringtobindingquarklist converts a string with one or more components to a binding list and a quark list.

For more information about the C language **Xlib** routine called by the **FXlib** binding routine **fxrmstringtobindingquarklist**, see the **XrmStringToBindingQuarkList** routine in "Xlib Functions."

¦ Copyright IBM Corp. 1985, 1991

3.5.280 fxrmstringtoquark

```
integer*4 fxrmstringtoquark
external  fxrmstringtoquark
character*256 string
integer*4 quark

quark = fxrmstringtoquark( string)
```

string Specifies the *string* for a *quark* to be allocated.

fxrmstringtoquark converts a string to a quark.

Return value *quark* contains the address of the *quark* allocated. If no *quark* is allocated, it contains **NULL**.

For more information about the C language **Xlib** routine called by the **FXlib** binding routine **fxrmstringtoquark**, see the **XrmStringToQuark** routine in "Xlib Functions."

¦ Copyright IBM Corp. 1985, 1991

3.5.281 fxrmstringtoquarklist

external fxrmstringtoquarklist
character*256 *string*
integer*4 *quarksreturn*

call fxrmstringtoquarklist(*string, quarksreturn*)

string Specifies the string for a quark to be allocated.

quarksreturn Specifies the list of quarks.

fxrmstringtoquarklist converts a string with one or more components to a quark list.

For more information about the C language **Xlib** routine called by the **FXlib** binding routine **fxrmstringtoquarklist** see the **XrmStringToQuarkList** routine in "Xlib Functions."

¦ Copyright IBM Corp. 1985, 1991

3.5.282 fxrmuniquequark

```
integer*4 fxrmuniquequark
external  fxrmuniquequark
integer*4 quark

quark =  fxrmuniquequark()
```

fxrmuniquequark allocates a new quark.

Return value *quark* is a new quark of type **integer**.

For more information about the C language **Xlib** routine called by the **FXlib** binding routine **fxrmuniquequark**, see the **XrmUniqueQuark** routine in "Xlib Functions."

¦ Copyright IBM Corp. 1985, 1991

3.5.283 fxrootwindow

```
integer*4  fxrootwindow
external   fxrootwindow
integer*4  display
integer*4  screen
integer*4  rc

rc = fxrootwindow( display, screen)
```

display Specifies the connection to the X Server.

screen Specifies the screen.

fxrootwindow obtains the root window.

Return value *rc* is the root window.

For more information about the C language **Xlib** routine called by the **FXlib** binding routine **fxrootwindow**, see the **XRootWindow** routine in "Xlib Functions."

¦ Copyright IBM Corp. 1985, 1991

3.5.284 fxrootwindowofscreen

```
integer*4  fxrootwindowofscreen
external   fxrootwindowofscreen
integer*4  screen
integer*4  rc

rc = fxrootwindowofscreen( screen)
```

screen Specifies the address of type **Screen** of the screen of the display.

fxrootwindowofscreen returns the root window of the screen specified.

Return value *rc* is the root window of the screen specified.

For more information about the C language **Xlib** routine called by the **FXlib** binding routine **fxrootwindowofscreen**, see the **XRootWindowOfScreen** routine in "Xlib Functions."

¦ Copyright IBM Corp. 1985, 1991

3.5.285 fxrotatebuffers

external fxrotatebuffers
integer*4 *display, rotate*

call fxrotatebuffers(*display, rotate*)

display Specifies the connection to the X Server.

rotate Specifies how much to rotate the cut buffer.

fxrotatebuffers rotates the cut buffers.

This routine can generate the errors **BadWindow**, **BadAtom**, and **BadMatch**.

For more information about the C language **Xlib** routine called by the **FXlib** binding routine **fxrotatebuffers**, see the **XRotateBuffers** routine in "Xlib Functions."

¦ Copyright IBM Corp. 1985, 1991

3.5.286 fxrotatewindowproperties

```
external  fxrotatewindowproperties
integer*4 display
integer*4 window
integer*4 properties
integer*4 numprop, npositions

call fxrotatewindowproperties( display, window, properties,
*                                   numprop, npositions)
```

display Specifies the connection to the X Server.

window Specifies the window ID.

properties Specifies the array of properties to be rotated.

numprop Specifies the length of the properties array.

npositions Specifies the rotation amount.

fxrotatewindowproperties rotates properties in the properties array.

This routine can generate errors **BadAtom**, **BadMatch**, or **BadWindow**.

For more information about the C language **Xlib** routine called by the **FXlib** binding routine **fxrotatewindowproperties**, see the **XRotateWindowProperties** routine in "Xlib Functions."

¦ Copyright IBM Corp. 1985, 1991

3.5.287 fxsavecontext

```
integer*4 fxsavecontext
external  fxsavecontext
integer*4 display, window, context
integer*4 data
integer*4 status

status = fxsavecontext( display, window, context, data)
```

display Specifies the connection to the X Server.

window Specifies the window associated with the data.

context Specifies the context type to which the data belongs.

data Specifies the starting address of the data to be associated with the window and type.

fxsavecontext saves a data value that corresponds to a window and context type.

For more information about the C language **Xlib** routine called by the **FXlib** binding routine **fxsavecontext**, see the **XSaveContext** routine in "Xlib Functions."

¦ Copyright IBM Corp. 1985, 1991

3.5.288 fxscreencount

```
integer*4  fxscreencount
external   fxscreencount
integer*4  display
integer*4  rc

rc = fxscreencount( display)
```

display Specifies the connection to the X Server.

fxscreencount obtains the number of screens available.

Return value *rc* is the number of available screens.

For more information about the C language **Xlib** routine called by the **FXlib** binding routine **fxscreencount**, see the **XScreenCount** routine in "Xlib Functions."

¦ Copyright IBM Corp. 1985, 1991

3.5.289 fxscreenofdisplay

```
integer*4  fxscreenofdisplay
external   fxscreenofdisplay
integer*4  display
integer*4  screennumber
integer*4  scrdisplay

scrdisplay = fxscreenofdisplay( display, screennumber)
```

display Specifies the connection to the X Server.

screennumber Specifies the *screen* number.

fxscreenofdisplay returns the address of the screen specified.

Return value *scrdisplay* is an address of type **Screen** of the screen structure for the display and screen specified.

For more information about the C language **Xlib** routine called by the **FXlib** binding routine **fxscreenofdisplay**, see the **XScreenOfDisplay** routine in "Xlib Functions."

¦ Copyright IBM Corp. 1985, 1991

3.5.290 fxselectinput

external fxselectinput
integer*4 *display*
integer*4 *window*
integer*4 *eventmask*

call fxselectinput(*display, window, eventmask*)

display Specifies the connection to the X Server.

window Specifies the window ID.

eventmask Specifies the event mask. This mask is a bitwise inclusive OR of valid event mask bits.

fxselectinput selects the events to report to the client application.

This routine generates errors **BadWindow** and **BadValue**.

For more information about the C language **Xlib** routine called by the **FXlib** binding routine **fxselectinput**, see the **XSelectInput** routine in "Xlib Functions."

¦ Copyright IBM Corp. 1985, 1991

3.5.291 fxsendevent

external fxsendevent
integer*4 *display*
integer*4 *window, propogate, eventmask, eventsend*

call fxsendevent(*display, window, propogate, eventmask, eventsend*)

display Specifies the connection to the X Server.

window Specifies the window ID for the destination window of the event.

propogate Specifies a value of **TRUE** if propogation is desired, or **FALSE** if it is not.

eventmask Specifies the event mask. This mask is the bitwise inclusive OR of one of the valid event mask bits.

eventsend Specifies the destination address of type **XEvent** in the client data area of the event structure to send to the specified window. The client must allocate the data area and free the data area when it is no longer needed.

fxsendevent sends an event to a specified window.

The variable *window* can be set to the window ID, to **PointerWindow**, or to **InputFocus**.

This routine can generate event errors **BadWindow** and **BadValue**.

For more information about the C language **Xlib** routine called by the **FXlib** binding routine **fxsendevent**, see the **XSendEvent** routine in "Xlib Functions."

¦ Copyright IBM Corp. 1985, 1991

3.5.292 fxservervendor

```
character*256  fxservervendor
external       fxservervendor
integer*4      display
character*256  servervend

servervend = fxservervendor( display)
```

display Specifies the connection to the X Server.

fxservervendor obtains the string that provides identification about the owner of the X Server.

Return value *servervend* is the name (a string) of the server vendor.

For more information about the C language **Xlib** routine called by the **FXlib** binding routine **fxservervendor**, see the **XServerVendor** routine in "Xlib Functions."

¦ Copyright IBM Corp. 1985, 1991

3.5.293 fxsetaccesscontrol

external fxsetaccesscontrol
integer*4 *display, mode*

call fxsetaccesscontrol(*display, mode*)

display Specifies the connection to the X Server.

mode Specifies whether to change the access control to enable or disable.

fxsetaccesscontrol changes access control.

The variable *mode* can be set to **EnableAccess**, which enables host access control, or, it can be set to **DisableAccess**, which disables host access control.

This routine can generate the errors **BadAlloc** and **BadAccess**.

For more information about the C language **Xlib** routine called by the **FXlib** binding routine **fxsetaccesscontrol**, see the **XSetAccessControl** routine in "Xlib Functions."

¦ Copyright IBM Corp. 1985, 1991

3.5.294 fxsetafterfunction

```
integer*4 fxsetafterfunction
external  fxsetafterfunction
integer*4 display
integer*4 proc
integer*4 rc

rc = fxsetafterfunction( display, proc)
```

display Specifies the connection to the X Server.

proc Specifies the function to call after an **Xlib** function that generates a protocol request completes its work.

fxsetafterfunction sets which function to call.

Return value *rc* is the status of the request.

For more information about the C language **Xlib** routine called by the **FXlib** binding routine **fxsetafterfunction**, see the **XSetAfterFunction** routine in "Xlib Functions."

¦ Copyright IBM Corp. 1985, 1991

3.5.295 fxsetarcmode

external fxsetarcmode
integer*4 *display*
integer*4 *gc*
integer*4 *arcmode*

call fxsetarcmode(*display, gc, arcmode*)

display Specifies the connection to the X Server.

gc Specifies an address of type **GC** that has the graphics context.

arcmode Specifies the arc mode.

fxsetarcmode sets the arc mode in the specified graphics context.

Possible values for *arcmode* are **ArcChord**, which specifies that arcs be chord-filled, and **ArcPieSlice**, which specifies that arcs be pie-slice filled.

This function can generate the errors **BadGC** and **BadValue**.

For more information about the C language **Xlib** routine called by the **FXlib** binding routine **fxsetarcmode**, see the **XSetArcMode** routine in "Xlib Functions."

¦ Copyright IBM Corp. 1985, 1991

3.5.296 fxsetbackground

external fxsetbackground
integer*4 *display*
integer*4 *gc*
integer*4 *background*

call fxsetbackground(*display, gc, background*)

display Specifies the connection to the X Server.

gc Specifies the address of type **GC** that has the graphics context.

background Specifies the background to be set for the specified graphics context.

fxsetbackground sets the background in the specified graphics context.

This routine can generate the error **BadGC**.

For more information about the C language **Xlib** routine called by the **FXlib** binding routine **fxsetbackground**, see the **XSetBackground** routine in "Xlib Functions."

¦ Copyright IBM Corp. 1985, 1991

3.5.297 fxsetclasshint

external fxsetclasshint
integer*4 *display*
integer*4 *window, classhints*

call fxsetclasshint(*display, window, classhints*)

display Specifies the connection to the X Server.

window Specifies the window ID.

classhints Specifies an address of type **XClassHint** of the class hints.

fxsetclasshint sets the class hint of the specified window

This routine can generate the errors **BadWindow** and **BadAlloc**.

For more information about the C language **Xlib** routine called by the **FXlib** binding routine **fxsetclasshint**, see the **XSetClassHint** routine in "Xlib Functions."

¦ Copyright IBM Corp. 1985, 1991

3.5.298 fxsetclipmask

external fxsetclipmask
integer*4 *display*
integer*4 *gc*
integer*4 *pixmap*

call fxsetclipmask(*display, gc, pixmap*)

display Specifies the connection to the X Server.

gc Specifies an address of type **GC** that has the graphics context.

pixmap Specifies the pixmap.

fxsetclipmask sets the clipmask in the specified graphics context to the specified pixmap.

This routine can generate errors **BadGC**, **BadMatch**, and **BadValue**.

For more information about the C language **Xlib** routine called by the **FXlib** binding routine **fxsetclipmask**, see the **XSetClipMask** routine in "Xlib Functions."

¦ Copyright IBM Corp. 1985, 1991

3.5.299 fxsetcliporigin

external fxsetcliporigin
integer*4 *display*
integer*4 *gc*
integer*4 *clipxorigin, clipyorigin*

call fxsetcliporigin(*display, gc, clipxorigin, clipyorigin*)

display Specifies the connection to the X Server.

gc Specifies an address of type **GC** that has the graphics context.

clipxorigin Specifies the *x* coordinate of the clip origin.

clipyorigin Specifies the *y* coordinates of the clip origin.

fxsetcliporigin sets the clip origin in the specified graphics context.

This routine can generate the error **BadGC**.

For more information about the C language **Xlib** routine called by the **FXlib** binding routine **fxsetcliporigin**, see the **XSetClipOrigin** routine in "Xlib Functions."

¦ Copyright IBM Corp. 1985, 1991

3.5.300 fxsetcliprectangles

```
external  fxsetcliprectangles
integer*4 display
integer*4 gc
integer*4 clipxorigin, clipyorigin
integer*4 rectangles
integer*4 n, ordering

call fxsetcliprectangles( display, gc, clipxorigin,
*                            clipyorigin, rectangles, n, ordering)
```

display Specifies the connection to the X Server.

gc Specifies an address of type **GC** that has the graphics context.

clipxorigin Specifies the *x* coordinate of the clip origin.

clipyorigin Specifies the *y* coordinate of the clip origin.

rectangles Specifies an address of **XRectangle** that has an array of rectangles. These are the rectangles to be specified in the graphics context. Each **XRectangle** structure occupies 8 bytes.

n Specifies the number of rectangles.

ordering Specifies the ordering relations on the rectangles.

fxsetcliprectangles changes the clipmask in the specified graphics context to the specified list of rectangles.

Possible values for the variable *ordering* are **Unsorted**, **YSorted**, **YXSorted**, or **YXBanded**.

This routine can generate errors **BadGC**, **BadAlloc**, and **BadValue**.

For more information about the C language **Xlib** routine called by the **FXlib** binding routine **fxsetcliprectangles**, see the **XSetClipRectangles** routine in "Xlib Functions."

¦ Copyright IBM Corp. 1985, 1991

3.5.301 fxsetclosedownmode

external fxsetclosedownmode
integer*4 *display*
integer*4 *closemode*

call fxsetclosedownmode(*display, closemode*)

display Specifies the connection to the X Server.

closemode Specifies the client close-down mode to be changed.

fxsetclosedownmode changes the close-down mode of a client.

The variable *closemode* can be set to **DestroyAll**, **RetainPermanent**, or **RetainTemporary**.

This routine can generate the error **BadValue**.

For more information about the C language **Xlib** routine called by the **FXlib** binding routine **fxsetclosedownmode**, see the **XSetCloseDownMode** routine in "Xlib Functions."

¦ Copyright IBM Corp. 1985, 1991

3.5.302 fxsetcommand

external fxsetcommand
integer*4 *display*
integer*4 *window*
integer*4 *argv, argc*

call fxsetcommand(*display, window, argv, argc*)

display Specifies the connection to the X Server.

window Specifies the window ID.

argv Specifies the starting address of a list of command and arguments used to start the application.

argc Specifies the number of arguments.

fxsetcommand sets the value of the command atom by setting the **WM_COMMAND** property.

This routine can generate the errors **BadWindow** and **BadAlloc**.

For more information about the C language **Xlib** routine called by the **FXlib** binding routine **fxsetcommand**, see the **XSetCommand** routine in "Xlib Functions."

¦ Copyright IBM Corp. 1985, 1991

3.5.303 fxsetdashes

external fxsetdashes
integer*4 *display*
integer*4 *gc*
integer*4 *dashoffset, dashlist, n*

call fxsetdashes(*display, gc, dashoffset, dashlist, n*)

display Specifies the connection to the X Server.

gc Specifies address of type **GC** that has the graphics context.

dashoffset Specifies the phase of the pattern for the dashed line style to be set for the specified graphics context.

dashlist Specifies the dash list for the dashed line style to be set for the specified graphics context. The first byte of each 2-byte integer corresponds to the even dash and the second byte to the odd dash. The client must allocate the dash list and must free the dash list when it is no longer needed.

n Specifies the length of the dash list argument.

fxsetdashes sets the *dashoffset* and *dashlist* for dashed line styles in the specified graphics context.

This routine can generate the errors **BadValue**, **BadGC**, and **BadAlloc**.

For more information about the C language **Xlib** routine called by the **FXlib** binding routine **fxsetdashes**, see the **XSetDashes** routine in "Xlib Functions."

¦ Copyright IBM Corp. 1985, 1991

3.5.304 fxseterrorhandler

external fxseterrorhandler
integer*4 *handler*

call fxseterrorhandler(*handler*)

handler Specifies the user supplied program error handler function.

fxseterrorhandler sets the event error handler to the specified function. The function parameters are the connection to the X Server and an address of type **XErrorEvent** which provides the error reporting parameters.

For more information about the C language **Xlib** routine called by the **FXlib** binding routine **fxseterrorhandler**, see the **XSetErrorHandler** routine in "Xlib Functions."

¦ Copyright IBM Corp. 1985, 1991

3.5.305 fxsetfillrule

external fxsetfillrule
integer*4 *display*
integer*4 *gc*
integer*4 *fillrule*

call fxsetfillrule(*display, gc, fillrule*)

display Specifies the connection to the X Server.

gc Specifies the address of type **GC** that has the graphics context.

fillrule Specifies the fill rule to be set for the specified graphics context.

fxsetfillrule sets the fill rule in the specified graphics context.

Possible values for the variable *fillrule* are **EvenOddRule** or **WindingRule**.

This routine can generate errors **BadValue** and **BadGC**.

For more information about the C language **Xlib** routine called by the **FXlib** binding routine **fxsetfillrule**, see the **XSetFillRule** routine in "Xlib Functions."

¦ Copyright IBM Corp. 1985, 1991

3.5.306 fxsetfillstyle

external fxsetfillstyle
integer*4 *display*
integer*4 *gc*
integer*4 *fillstyle*

call fxsetfillstyle(*display, gc, fillstyle*)

display	Specifies the connection to the X Server.
gc	Specifies the address of **GC** that has the graphics context.
fillstyle	Specifies the fill style to be set for the specified graphics context.

fxsetfillstyle sets the fill style in the specified graphics context.

Possible values for *fillstyle* are **FillSolid**, **FillTiled**, **FillStippled**, or **FillOpaqueStippled**.

This routine can generate the errors **BadGC**, and **BadValue**.

For more information about the C language **Xlib** routine called by the **FXlib** binding routine **fxsetfillstyle**, see the **XSetFillStyle** routine in "Xlib Functions."

¦ Copyright IBM Corp. 1985, 1991

3.5.307 fxsetfont

external fxsetfont
integer*4 *display*
integer*4 *gc*
integer*4 *font*

call fxsetfont(*display, gc, font*)

display Specifies the connection to the X Server.

gc Specifies an address of type **GC** that has the graphics context.

font Specifies the font ID.

fxsetfont sets the current font in the specified graphics context.

This routine can generate the errors **BadGC**, **BadFont**, and **BadAlloc**.

For more information about the C language **Xlib** routine called by the **FXlib** binding routine **fxsetfont**, see the **XSetFont** routine in "Xlib Functions."

¦ Copyright IBM Corp. 1985, 1991

3.5.308 fxsetfontpath

external fxsetfontpath
integer*4 *display*
integer*4 *directories*
integer*4 *ndirs*

call fxsetfontpath(*display, directories, ndirs*)

display Specifies the connection to the X Server.

directories Specifies the starting address of a list of addresses that contain the directory paths used to look for a font. Setting the path to the empty list restores the default path defined for the X Server. The client must allocate the directory list and must free the directory list when it is no longer needed.

ndirs Specifies the number of directories in the path.

fxsetfontpath sets the font search path.

This routine can generate the error **BadValue**.

For more information about the C language **Xlib** routine called by the **FXlib** binding routine **fxsetfontpath**, see the **XSetFontPath** routine in "Xlib Functions."

¦ Copyright IBM Corp. 1985, 1991

3.5.309 fxsetforeground

external fxsetforeground
integer*4 *display*
integer*4 *gc*
integer*4 *foreground*

call fxsetforeground(*display, gc, foreground*)

display Specifies the connection to the X Server.

gc Specifies the address of type **GC** that has the graphics context.

foreground Specifies the foreground to be set for the specified graphics context.

fxsetforeground sets the foreground in the specified graphics context.

This routine can generate error **BadGC**.

For more information about the C language **Xlib** routine called by the **FXlib** binding routine **fxsetforeground**, see the **XSetForeground** routine in "Xlib Functions."

¦ Copyright IBM Corp. 1985, 1991

3.5.310 fxsetfunction

external fxsetfunction
integer*4 *display*
integer*4 *gc*
integer*4 *function*

call fxsetfunction(*display, gc, function*)

display Specifies the connection to the X Server.

gc Specifies the address of type **GC** that has the graphics context.

function Specifies the function to be set for the specified graphics context.

fxsetfunction sets a specified value in the specified graphics context.

This routine can generate errors **BadGC** and **BadValue**.

For more information about the C language **Xlib** routine called by the **FXlib** binding routine **fxsetfunction**, see the **XSetFunction** routine in "Xlib Functions."

¦ Copyright IBM Corp. 1985, 1991

3.5.311 fxsetgraphicsexposures

```
external  fxsetgraphicsexposures
integer*4 display
integer*4 gc
integer*4 graphicsexposures

call fxsetgraphicsexposures( display, gc, graphicsexposures)
```

display	Specifies the connection to the X Server.
gc	Specifies an address of type **GC** that has the graphics context.
graphicsexposures	Specifies if **GraphicsExpose** events are to be reported when calling **fxcopyarea** and **fxcopyplane** with this graphics context.

fxsetgraphicsexposures sets the *graphicsexposures* flag in the specified graphics context. **GraphicsExpose** events are reported if **TRUE** is indicated. **GraphicsExpose** events are not reported if **FALSE** is indicated.

This routine can generate the errors **BadGC** and **BadValue**.

For more information about the C language **Xlib** routine called by the **FXlib** binding routine **fxsetgraphicsexposures**, see the **XSetGraphicsExposures** routine in "Xlib Functions."

¦ Copyright IBM Corp. 1985, 1991

3.5.312 fxseticonname

external fxseticonname
integer*4 *display*
integer*4 *window*
character*256 *iconname*

call fxseticonname(*display, window, iconname*)

display Specifies the connection to the X Server.

window Specifies the window ID of the window whose icon name is being set.

iconname Specifies the name to be displayed in the icon window. This name is returned by any subsequent call to **fxgeticonname**.

fxseticonname sets the name to be displayed in a window's icon.

This routine can generate the errors **BadWindow** and **BadAlloc**.

For more information about the C language **Xlib** routine called by the **FXlib** binding routine **fxseticonname**, see the **XSetIconName** routine in "Xlib Functions."

¦ Copyright IBM Corp. 1985, 1991

3.5.313 fxseticonsizes

external fxseticonsizes
integer*4 *display*
integer*4 *window, sizelist, count*

call fxseticonsizes(*display, window, sizelist, count*)

display Specifies the connection to the X Server.

window Specifies the window ID.

sizelist Specifies a starting address of type **XIconSize** of the icon size list. The client must allocate the size list and must free the size list when it is no longer needed.

count Specifies the number of items in the size list.

fxseticonsizes sets the value of the icon size atom.

This routine can generate the errors **BadWindow** and **BadAlloc**.

For more information about the C language **Xlib** routine called by the **FXlib** binding routine **fxseticonsizes**, see the **XSetIconSizes** routine in "Xlib Functions."

¦ Copyright IBM Corp. 1985, 1991

3.5.314 fxsetinputfocus

external fxsetinputfocus
integer*4 *display*
integer*4 *focus, revertto, time*

call fxsetinputfocus(*display, focus, revertto, time*)

display Specifies the connection to the X Server.

focus Specifies the destination window ID for the input focus.

revertto Specifies which window the input focus reverts to if the window becomes unviewable.

time Specifies the time in a time stamp in milliseconds or **CurrentTime**.

fxsetinputfocus sets the input focus.

The variable *focus* can be set to the window ID or **PointerRoot** or **None**.

The variable *revertto* can be set to **RevertToParent**, **RevertToPointerRoot**, or **RevertToNone**.

fxsetinputfocus can generate the errors **BadWindow** and **BadValue**.

For more information about the C language **Xlib** routine called by the **FXlib** binding routine **fxsetinputfocus**, see the **XSetInputFocus** routine in "Xlib Functions."

¦ Copyright IBM Corp. 1985, 1991

3.5.315 fxsetioerrorhandler

external fxsetioerrorhandler
integer*4 *handler*

call fxsetioerrorhandler(*handler*)

handler Specifies the user supplied program error handler. The function parameter is the connection to the X Server.

fxsetioerrorhandler sets the fatal I/O error handler to the specified function.

For more information about the C language **Xlib** routine called by the **FXlib** binding routine **fxsetioerrorhandler**, see the **XSetIOErrorHandler** routine in "Xlib Functions."

¦ Copyright IBM Corp. 1985, 1991

3.5.316 fxsetlineattributes

```
external  fxsetlineattributes
integer*4 display
integer*4 gc
integer*4 linewidth, linestyle
integer*4 capstyle, joinstyle

call fxsetlineattributes( display, gc, linewidth, linestyle,
*                         capstyle, joinstyle)
```

display Specifies the connection to the X Server.

gc Specifies the address of type **GC** that has the graphics context.

linewidth Specifies the line width to be set for the specified graphics context.

linestyle Specifies the line style to be set for the specified graphics context.

capstyle Specifies the line and line-cap style to be set for the specified graphics context.

joinstyle Specifies the line-join style to be set for the specified graphics context.

fxsetlineattributes sets the line drawing components in the specified graphics context.

Possible values for the variable *linestyle* are **LineSolid**, **LineOnOffDash**, or **LineDoubleDash**.

Possible values for the variable *capstyle* are **CapNotLast**, **CapButt**, **CapRound**, or **CapProjecting**.

Possible values for the variable *joinstyle* are **JoinMiter**, **JoinRound**, or **JoinBevel**.

This routine can generate errors **BadGC** and **BadValue**.

For more information about the C language **Xlib** routine called by the **FXlib** binding routine **fxsetlineattributes**, see the **XSetLineAttributes** routine in "Xlib Functions."

¦ Copyright IBM Corp. 1985, 1991

3.5.317 fxsetmodifiermapping

integer*4 fxsetmodifiermapping
external fxsetmodifiermapping
integer*4 *display, modmap*
integer*4 *rc*

rc = **fxsetmodifiermapping**(*display, modmap*)

display Specifies the connection to the X Server.

modmap Specifies the address of type **XModifierKeymap** of the keymap modifier data structure.

fxsetmodifiermapping sets those keycodes to be used as modifiers.

Return value *rc* indicates the status of the mapping modification.

This routine can generate the errors **BadValue** and **BadAlloc**.

For more information about the C language **Xlib** routine called by the **FXlib** binding routine **fxsetmodifiermapping**, see the **XSetModifierMapping** routine in "Xlib Functions."

¦ Copyright IBM Corp. 1985, 1991

3.5.318 fxsetnormalhints

external fxsetnormalhints
integer*4 *display*
integer*4 *window*
integer*4 *hints*

call fxsetnormalhints(*display, window, hints*)

display Specifies the connection to the X Server.

window Specifies the window ID.

hints Specifies an address of type **XSizeHints** that contains the sizing hints for a window in its normal state.

fxsetnormalhints sets the size hints for a window in its normal state.

This routine can generate the errors **BadWindow** and **BadAlloc**.

For more information about the C language **Xlib** routine called by the **FXlib** binding routine **fxsetnormalhints**, see the **XSetNormalHints** routine in "Xlib Functions."

¦ Copyright IBM Corp. 1985, 1991

3.5.319 fxsetplanemask

external fxsetplanemask
integer*4 *display*
integer*4 *gc*
integer*4 *planemask*

call fxsetplanemask(*display, gc, planemask*)

display Specifies the connection to the X Server.

gc Specifies the address of type **GC** that has the graphics context.

planemask Specifies the plane mask to be set for the specified graphics context.

fxsetplanemask sets the plane mask in the specified graphics context.

This routine can generate the error **BadGC**.

For more information about the C language **Xlib** routine called by the **FXlib** binding routine **fxsetplanemask**, see the **XSetPlaneMask** routine in "Xlib Functions."

¦ Copyright IBM Corp. 1985, 1991

3.5.320 fxsetpointermapping

```
integer*4 fxsetpointermapping
external fxsetpointermapping
integer*4 display
integer*4 map
integer*4 nmap
integer*4 rc

rc = fxsetpointermapping( display, map, nmap)
```

display Specifies the connection to the X Server.

map Specifies the starting address of a list of the current pointer mapping. The list is an array of byte values. The client must allocate the mapping list and must free the mapping list when it is no longer needed.

nmap Specifies the number of items in mapping list.

fxsetpointermapping sets the mapping of buttons on the pointer.

For more information about the C language **Xlib** routine called by the **FXlib** binding routine **fxsetpointermapping**, see the **XSetPointerMapping** routine in "Xlib Functions."

¦ Copyright IBM Corp. 1985, 1991

3.5.321 fxsetregion

external fxsetregion
integer*4 *display*
integer*4 *gc, r*

call fxsetregion(*display, gc, r*)

display Specifies the connection to the X Server.

gc Specifies the address of type **GC** that contains the graphics context.

r Specifies the address of type **_XRegion** of the region structure. This is the region where the specified graphics context is located.

fxsetregion sets the graphics contexts to the specified region.

For more information about the C language **Xlib** routine called by the **FXlib** binding routine **fxsetregion**, see the **XSetRegion** routine in "Xlib Functions."

¦ Copyright IBM Corp. 1985, 1991

3.5.322 fxsetscreensaver

external fxsetscreensaver
integer*4 *display*
integer*4 *timeout, interval*
integer*4 *preferblanking, allowexposures*

call fxsetscreensaver(*display, timeout, interval,*
** preferblanking, allowexposures*)

display	Specifies the connection to the X Server.
timeout	Specifies the timeout, in seconds, until the screen saver turns on.
interval	Specifies the interval between screen saver invocations.
preferblanking	Specifies whether to enable screen-blanking.
allowexposure	Specifies the current screen save control values.

fxsetscreensaver sets the screen saver.

The variable *preferblanking* can be set to **DontPreferBlanking**, **PreferBlanking**, or **DefaultBlanking**.

The variable *allowexposure* can be set to **DontAllowExposures**, **AllowExposures**, or **DefaultExposures**.

fxsetscreensaver can generate the error **BadValue**.

For more information about the C language **Xlib** routine called by the **FXlib** binding routine **fxsetscreensaver**, see the **XSetScreenSaver** routine in "Xlib Functions."

¦ Copyright IBM Corp. 1985, 1991

3.5.323 fxsetselectionowner

external fxsetselectionowner
integer*4 *display*
integer*4 *selection*
integer*4 *owner*
integer*4 *time*

call fxsetselectionowner(*display, selection, owner, time*)

display Specifies the connection to the X Server.

selection Specifies the selection atom.

owner Specifies the owner of the specified selection atom. This value should be set to a window ID or to **None**.

time Specifies the time in a timestamp in milliseconds or **CurrentTime**.

fxsetselectionowner sets the selection owner.

This routine can generate errors **BadAtom** and **BadWindow**.

For more information about the C language **Xlib** routine called by the **FXlib** binding routine **fxsetselectionowner**, see the **XSetSelectionOwner** routine in "Xlib Functions."

¦ Copyright IBM Corp. 1985, 1991

3.5.324 fxsetsizehints

external fxsetsizehints
integer*4 *display*
integer*4 *window, hints, property*
call fxsetsizehints (*display, window, hints, property*)

display Specifies the connection to the X Server.

window Specifies the window ID.

hints Specifies an address of type **XSizeHints** that contains the size hints.

property Specifies the property atom.

fxsetsizehints sets the size hints for the specified property in the specified window.

This routine can generate the errors **BadWindow**, **BadAlloc**, and **BadAtom**.

For more information about the C language **Xlib** routine called by the **FXlib** binding routine **fxsetsizehints**, see the **XSetSizeHints** routine in "Xlib Functions."

¦ Copyright IBM Corp. 1985, 1991

3.5.325 fxsetstandardcolormap

external fxsetstandardcolormap
integer*4 *display*
integer*4 *window, cmapreturn, property*
call **fxsetstandardcolormap**(*display, window, cmapreturn, property*)

display Specifies the connection to the X Server.

window Specifies the window ID.

cmapreturn Returns the address of the colormap associated with the specified atom.

property Specifies the property atom of the colormap.

fxsetstandardcolormap creates or changes a standard colormap.

This routine can generate the errors **BadWindow**, **BadAtom**, and **BadAlloc**.

For more information about the C language **Xlib** routine called by the **FXlib** binding routine **fxsetstandardcolormap**, see the **XSetStandardColormap** routine in "Xlib Functions."

¦ Copyright IBM Corp. 1985, 1991

3.5.326 fxsetstandardproperties

external fxsetstandardproperties
integer*4 *display*
integer*4 *window*
character*256 *windowname*
character*256 *iconname*
integer*4 *iconpixmap, argv, argc, hints*

call fxsetstandardproperties(*display, window, windowname, iconname,*
** iconpixmap, argv, argc, hints*)

display Specifies the connection to the X Server.

window Specifies the window ID.

windowname Specifies the name of the window.

iconname Specifies the name to be displayed in the window's icon.

iconpixmap Specifies the pixmap to be used for the icon or **None**.

argv Specifies the starting address of the list of command and arguments used to start the application.

argc Specifies the number of arguments.

hints Specifies the address of type **XSizeHints** of the sizing hints for the window in its normal state.

fxsetstandardproperties specifies a minimum set of properties for an application.

This routine can generate event errors **BadWindow** and **BadAlloc**.

For more information about the C language **Xlib** routine called by the **FXlib** binding routine **fxsetstandardproperties**, see the **XSetStandardProperties** routine in "Xlib Functions."

¦ Copyright IBM Corp. 1985, 1991

3.5.327 fxsetstate

```
external  fxsetstate
integer*4 display
integer*4 gc
integer*4 foreground, background
integer*4 function, planemask

call fxsetstate( display, gc, foreground, background,
*                 function, planemask)
```

display Specifies the connection to the X Server.

gc Specifies the address of type **GC** that has the graphics context.

foreground Specifies the foreground to be set for the specified graphics context.

background Specifies the background to be set for the specified graphics context.

function Specifies the function to be set for the specified graphics context.

planemask Specifies the plane mask to be set for the specified graphics context.

fxsetstate sets the foreground, background, plane mask, and function components for the specified graphics context.

This routine can generate the errors **BadGC** and **BadValue**.

For more information about the C language **Xlib** routine called by the **FXlib** binding routine **fxsetstate**, see the **XSetState** routine in "Xlib Functions."

¦ Copyright IBM Corp. 1985, 1991

3.5.328 fxsetstipple

external fxsetstipple
integer*4 *display*
integer*4 *gc*
integer*4 *stipple*

call fxsetstipple(*display, gc, stipple*)

display Specifies the connection to the X Server.

gc Specifies an address of type **GC** that has the graphics context.

stipple Specifies the stipple to be set for the specified graphics context.

fxsetstipple sets the stipple in the specified graphics context.

This routine can generate errors **BadGC**, **BadPixmap**, **BadMatch** and **BadAlloc**.

For more information about the C language **Xlib** routine called by the **FXlib** binding routine **fxsetstipple**, see the **XSetStipple** routine in "Xlib Functions."

¦ Copyright IBM Corp. 1985, 1991

3.5.329 fxsetsubwindowmode

external fxsetsubwindowmode
integer*4 *display*
integer*4 *gc*
integer*4 *subwindowmode*

call fxsetsubwindowmode(*display, gc, subwindowmode*)

display	Specifies the connection to the X Server.
gc	Specifies an address of type **GC** that has the graphics context.
subwindowmode	Specifies the subwindow mode.

fxsetsubwindowmode sets the subwindow mode in the specified graphics context.

Possible values for the variable *subwindowmode* are **ClipByChildren**, which clips source and destination by all viewable children, and **IncludeInferiors**, which draws through all subwindows and does not clip by inferiors.

This routine can generate the errors **BadGC** and **BadValue**.

For more information about the C language **Xlib** routine called by the **FXlib** binding routine **fxsetsubwindowmode**, see the **XSetSubwindowMode** routine in "Xlib Functions."

¦ Copyright IBM Corp. 1985, 1991

3.5.330 fxsettile

external fxsettile
integer*4 *display*
integer*4 *gc*
integer*4 *tile*

call fxsettile(*display, gc, tile*)

display Specifies the connection to the X Server.

gc Specifies an address of type **GC** that has the graphics context.

tile Specifies the fill tile to be set for the specified graphics context.

fxsettile sets the fill tile in the specified graphics context.

This routine can generate the errors **BadGC**, **BadPixmap**, **BadMatch**, and **BadAlloc**.

For more information about the C language **Xlib** routine called by the **FXlib** binding routine **fxsettile**, see the **XSetTile** routine in "Xlib Functions."

¦ Copyright IBM Corp. 1985, 1991

3.5.331 fxsettransientforhint

```
external fxsettransientforhint
integer*4 display
integer*4 window, propwindow

call fxsettransientforhint( display, window, propwindow)
```

display Specifies the connection to the X Server.

window Specifies the window ID.

propwindow Specifies the window ID that the **WM_TRANSIENT_FOR** property is to be set to.

fxsettransientforhint sets the **WM_TRANSIENT_FOR** property for a window.

This routine can generate the errors **BadWindow** and **BadAlloc**.

For more information about the C language **Xlib** routine called by the **FXlib** binding routine **fxsettransientforhint**, see the **XSetTransientForHint** routine in "Xlib Functions."

¦ Copyright IBM Corp. 1985, 1991

3.5.332 fxsettsorigin

external fxsettsorigin
integer*4 *display*
integer*4 *gc*
integer*4 *tsxorigin, tsyorigin*

call fxsettsorigin(*display, gc, tsxorigin, tsyorigin*)

display Specifies the connection to the X Server.

gc Specifies an address of type **GC** that has the graphics context.

tsxorigin Specifies the *x* coordinate of the tile or stipple origin.

tsyorigin Specifies the *y* coordinate of the tile or stipple origin.

fxsettsorigin sets the tile or stipple origin in the specified graphics context.

This routine can generate error **BadGC**.

For more information about the C language **Xlib** routine called by the **FXlib** binding routine **fxsettsorigin**, see the **XSetTSOrigin** routine in "Xlib Functions."

¦ Copyright IBM Corp. 1985, 1991

3.5.333 fxsetwindowbackground

external fxsetwindowbackground
integer*4 *display*
integer*4 *window*
integer*4 *backgroundpixel*

call fxsetwindowbackground(*display, window, backgroundpixel*)

display	Specifies the connection to the X Server.
window	Specifies the window ID.
backgroundpixel	Specifies the pixel value of the background. This value determines which entry in the colormap is used.

fxsetwindowbackground sets the background of a specified window to the pixel specified.

For more information about the C language **Xlib** routine called by the **FXlib** binding routine **fxsetwindowbackground**, see the **XSetWindowBackground** routine in "Xlib Functions."

¦ Copyright IBM Corp. 1985, 1991

3.5.334 fxsetwindowbackgroundpixmap

external fxsetwindowbackgroundpixmap
integer*4 *display*
integer*4 *window*
integer*4 *backgroundpixmap*

call fxsetwindowbackgroundpixmap(*display, window, backgroundpixmap*)

display	Specifies the connection to the X Server.
window	Specifies the window ID.
backgroundpixmap	Specifies the background pixmap.

fxsetwindowbackgroundpixmap sets the background of a specified window to the pixmap specified. If a pixmap ID is specified in *backgroundpixmap*, the background is painted with this pixmap. If **None** is specified, no background is painted. If **ParentRelative** is specified, the pixmap of the parent window is used.

For more information about the C language **Xlib** routine called by the **FXlib** binding routine **fxsetwindowbackgroundpixmap**, see the **XSetWindowBackgroundPixmap** routine in "Xlib Functions."

¦ Copyright IBM Corp. 1985, 1991

3.5.335 fxsetwindowborder

external fxsetwindowborder
integer*4 *display*
integer*4 *window*
integer*4 *borderpixel*

call fxsetwindowborder(*display, window, borderpixel*)

display Specifies the connection to the X Server.

window Specifies the window ID.

borderpixel Specifies the entry in the colormap.

fxsetwindowborder changes and repaints the border of the window to the pixel specified.

For more information about the C language **Xlib** routine called by the **FXlib** binding routine **fxsetwindowborder**, see the **XSetWindowBorder** routine in "Xlib Functions."

¦ Copyright IBM Corp. 1985, 1991

3.5.336 fxsetwindowborderpixmap

```
external  fxsetwindowborderpixmap
integer*4 display
integer*4 window
integer*4 borderpixmap
call fxsetwindowborderpixmap( display, window, borderpixmap)
```

display Specifies the connection to the X Server.

window Specifies the window ID.

borderpixmap Specifies the border pixmap.

fxsetwindowborderpixmap changes and repaints the border tile of a window. If a pixmap ID is specified in *borderpixmap*, the associated pixmap is used for the border. If **CopyFromParent** is specified, a copy of the border of the parent window pixmap is used.

For more information about the C language **Xlib** routine called by the **FXlib** binding routine **fxsetwindowborderpixmap**, see the **XSetWindowBorderPixmap** routine in "Xlib Functions."

¦ Copyright IBM Corp. 1985, 1991

3.5.337 fxsetwindowborderwidth

```
external  fxsetwindowborderwidth
integer*4 display
integer*4 window
integer*4 width

call fxsetwindowborderwidth( display, window, width)
```

display Specifies the connection to the X Server.

window Specifies the window ID.

width Specifies the window border width.

fxsetwindowborderwidth changes the border width of a window.

For more information about the C language **Xlib** routine called by the **FXlib** binding routine **fxsetwindowborderwidth**, see the **XSetWindowBorderWidth** routine in "Xlib Functions."

¦ Copyright IBM Corp. 1985, 1991

3.5.338 fxsetwindowcolormap

```
external  fxsetwindowcolormap
integer*4 display
integer*4 window, cmap

call fxsetwindowcolormap( display, window, cmap)
```

display Specifies the connection to the X Server.

window Specifies the window identification for the window to set the colormap.

cmap Specifies the colormap identification.

fxsetwindowcolormap sets the colormap for the window specified.

This routine can generate errors **BadWindow**, **BadColor**, and **BadMatch**.

For more information about the C language **Xlib** routine called by the **FXlib** binding routine **fxsetwindowcolormap**, see the **XSetWindowColormap** routine in "Xlib Functions."

¦ Copyright IBM Corp. 1985, 1991

3.5.339 fxsetwmhints

external fxsetwmhints
integer*4 *display*
integer*4 *window*
integer*4 *wmhints*

call fxsetwmhints(*display, window, wmhints*)

display Specifies the connection to the X Server.

window Specifies the window ID.

wmhints Specifies the address type **XSetWMHints** that contains the window manager hints.

fxsetwmhints sets the value of the window manager hints atom.

This routine can generate the errors **BadWindow** and **BadAlloc**.

For more information about the C language **Xlib** routine called by the **FXlib** binding routine **fxsetwmhints**, see the **XSetWMHints** routine in "Xlib Functions."

¦ Copyright IBM Corp. 1985, 1991

3.5.340 fxsetzoomhints

external fxsetzoomhints
integer*4 *display*
integer*4 *window*
integer*4 *zhints*

call fxsetzoomhints(*display, window, zhints*)

display Specifies the connection to the X Server.

window Specifies the window ID.

zhints Specifies an address of type **XSizeHints** that contains the zoom hints.

fxsetzoomhints sets the value of the zoom hints atom.

This routine can generate the errors **BadWindow** and **BadAlloc**.

For more information about the C language **Xlib** routine called by the **FXlib** binding routine **fxsetzoomhints**, see the **XSetZoomHints** routine in the C language **Xlib** documentation.

¦ Copyright IBM Corp. 1985, 1991

3.5.341 fxshrinkregion

```
external  fxshrinkregion
integer*4 r, dx, dy

call fxshrinkregion( r, dx, dy)
```

r	Specifies the address of type **_XRegion** of the region structure.
dx, dy	Specify the *x* and *y* coordinates that define the amount by which to reduce the region.

fxshrinkregion reduces the specified region by a specified amount.

For more information about the C language **Xlib** routine called by the **FXlib** binding routine **fxshrinkregion**, see the **XShrinkRegion** routine in "Xlib Functions."

¦ Copyright IBM Corp. 1985, 1991

3.5.342 fxstorebuffer

external fxstorebuffer
integer*4 *display*
integer*4 *bytes*
integer*4 *nbytes, buffer*

call fxstorebuffer(*display, bytes, nbytes, buffer*)

display	Specifies the connection to the X Server.
bytes	Specifies the starting address of a list of bytes to be stored in the specified buffer. The bytes are not necessarily ASCII characters and the list is not necessarily null-terminated.
nbytes	Specifies the number of bytes of the argument to be stored.
buffer	Specifies the buffer in which to store the bytes.

fxstorebuffer stores data in a specified cut buffer.

This routine can generate errors **BadWindow, BadAtom**, and **BadAlloc**.

For more information about the C language **Xlib** routine called by the **FXlib** binding routine **fxstorebuffer**, see the **XStoreBuffer** routine in "Xlib Functions."

¦ Copyright IBM Corp. 1985, 1991

3.5.343 fxstorebytes

external fxstorebytes
integer*4 *display*
integer*4 *bytes*
integer*4 *nbytes*

call fxstorebytes(*display, bytes, nbytes*)

display Specifies the connection to the X Server.

bytes Specifies the address of a list of bytes to be stored in buffer zero. The bytes are not necessarily ASCII characters, and the list is not necessarily null-terminated.

nbytes Specify the number of bytes of the argument to be stored.

fxstorebytes stores data in cut buffer zero.

This routine can generate the errors **BadWindow** and **BadAlloc**.

For more information about the C language **Xlib** routine called by the **FXlib** binding routine **fxstorebytes**, see the **XStoreBytes** routine in "Xlib Functions."

¦ Copyright IBM Corp. 1985, 1991

3.5.344 fxstorecolor

external fxstorecolor
integer*4 *display*
integer*4 *cmap*
integer*4 *screendefreturn*

call fxstorecolor(*display, cmap, screendefreturn*)

display Specifies the connection to the X Server.

cmap Specifies the colormap ID.

screendefreturn Returns an address of type **XColor** that has the values used in the colormap.

fxstorecolor sets the color of the specified pixel value to the closest available hardware color.

This routine can generate error **BadColor**.

For more information about the C language **Xlib** routine called by the **FXlib** binding routine **fxstorecolor**, see the **XStoreColor** routine in "Xlib Functions."

¦ Copyright IBM Corp. 1985, 1991

3.5.345 fxstorecolors

external fxstorecolors
integer*4 *display*
integer*4 *cmap*
integer*4 *defs*, *ncolors*

call fxstorecolors(*display, cmap, defs, ncolors*)

display Specifies the connection to the X Server.

cmap Specifies the colormap ID.

defs Specifies an address of an array of color definitions of type **XColor**. Each color definition occupies 12 bytes of data.

ncolors Specifies the number of type **XColor** data structures in the color definition array.

fxstorecolors sets the colors of the specified pixel values to the closest available hardware colors.

Errors that can be generated by **fxstorecolors** are **BadAccess** and **BadColor**.

For more information about the C language **Xlib** routine called by the **FXlib** binding routine **fxstorecolors**, see the **XStoreColors** routine in "Xlib Functions."

¦ Copyright IBM Corp. 1985, 1991

3.5.346 fxstorename

external fxstorename
integer*4 *display*
integer*4 *window*
character*256 *windowname*

call fxstorename(*display, window, windowname*)

display Specifies the connection to the X Server.

window Specifies the ID of the window whose icon name is being set.

windowname Specifies the name of the window. The name is returned by any subsequent call to **fxfetchname**.

fxstorename sets the name of a window.

This routine can generate the errors **BadWindow** and **BadAlloc**.

For more information about the C language **Xlib** routine called by the **FXlib** binding routine **fxstorename**, see the **XStoreName** routine in "Xlib Functions."

¦ Copyright IBM Corp. 1985, 1991

3.5.347 fxstorenamedcolor

external fxstorenamedcolor
integer*4 *display*
integer*4 *cmap*
character*256 *color*
integer*4 *pixel, flags*

call fxstorenamedcolor(*display, cmap, color, pixel, flags*)

display Specifies the connection to the X Server.

cmap Specifies the colormap ID.

color Specifies the color-name string, for example *red*. The function then allocates this color cell.

pixel Specifies the entry in the colormap.

flags Specifies which red, green, and blue indexes are set.

fxstorenamedcolor allocates a color cell by name.

Errors that this routine can generate are **BadColor, BadAccess, BadName,** and **BadValue**.

For more information about the C language **Xlib** routine called by the **FXlib** binding routine **fxstorenamedcolor**, see the **XStoreNamedColor** routine in "Xlib Functions."

¦ Copyright IBM Corp. 1985, 1991

3.5.348 fxstringtokeysym

integer*4 fxstringtokeysym
external fxstringtokeysym
character*256 *string*
integer*4 *status*

status = **fxstringtokeysym**(*string*)

string Specifies the string name of the keysym to be converted.

fxstringtokeysym converts the keysym name to the keysym code.

The return value *status* returns **NoSymbol** if the specified string does not match a valid keysym.

For more information about the C language **Xlib** routine called by the **FXlib** binding routine **fxstringtokeysym**, see the **XStringToKeysym** routine in "Xlib Functions."

¦ Copyright IBM Corp. 1985, 1991

3.5.349 fxsubimage

```
integer*4 fxsubimage
external  fxsubimage
integer*4 ximage, x, y, subimagewidth, subimageheight
integer*4 subimage

subimage = fxsubimage( ximage, x, y, subimagewidth, subimageheight)
```

ximage	Specifies the address of type **XImage** of the image.
x, y	Specify the *x* and *y* coordinates.
subimagewidth	Specifies the *width* of the new subimage in pixels.
subimageheight	Specifies the *height* of the new subimage in pixels.

fxsubimage creates a subimage.

The return value *subimage* is the address of type **XImage** of the subimage requested.

For more information about the C language **Xlib** routine called by the **FXlib** binding routine **fxsubimage**, see the **XSubImage** routine in "Xlib Functions."

¦ Copyright IBM Corp. 1985, 1991

3.5.350 fxsubtractregion

external fxsubtractregion
integer*4 *sra, srb, dr*

call fxsubtractregion(*sra, srb, dr*)

sra, srb Specify the addresses of type **_XRegion** of the two region structures with which to perform the computation.

dr Specifies the address of type **_XRegion** of the resulting region structure.

fxsubtractregion subtracts two regions.

For more information about the C language **Xlib** routine called by the **FXlib** binding routine **fxsubtractregion**, see the **XSubtractRegion** routine in "Xlib Functions."

¦ Copyright IBM Corp. 1985, 1991

3.5.351 fxsync

external fxsync
integer*4 *display, discard*

call fxsync(*display, discard*)

display Specifies the connection to the X Server.

discard Specifies whether events on the event queue are discarded. If *discard* is zero, no events are discarded. If *discard* is one, all events are discarded.

fxsync flushes the output buffer and then waits until all requests have been processed.

For more information about the C language **Xlib** routine called by the **FXlib** binding routine **fxsync**, see the **XSync** routine in "Xlib Functions."

¦ Copyright IBM Corp. 1985, 1991

3.5.352 fxsynchronize

```
integer*4 fxsynchronize
external  fxsynchronize
integer*4 display
integer*4 onoff
integer*4 rc

rc = fxsynchronize( display, onoff)
```

display Specifies the connection to the X Server.

onoff Specifies whether to enable or disable synchronization. If *onoff* is zero, synchronization is disabled. If *onoff* is one, synchronization is enabled.

fxsynchronize enables or disables synchronization.

For more information about the C language **Xlib** routine called by the **FXlib** binding routine **fxsynchronize**, see the **XSynchronize** routine in "Xlib Functions."

¦ Copyright IBM Corp. 1985, 1991

3.5.353 fxtextextents

```
external  fxtextextents
integer*4 fontstruct
character*256  string
integer*4 ncharacters, directionreturn, ascentreturn
integer*4 descentreturn, overallreturn

call fxtextextents( fontstruct, string, ncharacters,
*                       directionreturn, ascentreturn, descentreturn, overallr
```

fontstruct Specifies the address of type **XFontStruct** that has the font information used for the width computation.

string Specifies the character string of 8-bit characters.

ncharacters Specifies the number of characters in the character string.

directionreturn Returns the value of the direction hint member **FontLeftToRight** or **FontRightToLeft**.

ascentreturn Returns the ascent member.

descentreturn Returns the descent member.

overallreturn Returns the address of type **XCharStruct** that has the overall size in the specified data structure.

fxtextextents determines the logical extents of the specified 8-bit character string.

For more information about the C language **Xlib** routine called by the **FXlib** binding routine **fxtextextents**, see the **XTextExtents** routine in "Xlib Functions."

¦ Copyright IBM Corp. 1985, 1991

3.5.354 fxtextextents16

```
external  fxtextextents16
integer*4 fontstruct
integer*4 string
integer*4 ncharacters, directionreturn, ascentreturn
integer*4 descentreturn, overallreturn

call fxtextextents16( fontstruct, string, ncharacters,
*                         directionreturn, ascentreturn, descentreturn, overa
```

fontstruct Specifies the address of type **XFontStruct** that has the font information used for the width computation.

string Specifies the address of type **XChar2b** that has a character string of 16-bit characters.

ncharacters Specifies the number of characters in the character string.

directionreturn Returns the value of the direction hint member **FontLeftToRight** or **FontRightToLeft**.

ascentreturn Returns the ascent member.

descentreturn Returns the descent member.

overallreturn Returns the address of type **XCharStruct** that has the overall size in the specified data structure.

fxtextextents16 determines the logical extents of the specified 16-bit character string.

For more information about the C language **Xlib** routine called by the **FXlib** binding routine **fxtextextents16**, see the **XTextExtents16** routine in "Xlib Functions."

¦ Copyright IBM Corp. 1985, 1991

3.5.355 fxtextwidth

```
integer*4  fxtextwidth
external  fxtextwidth
integer*4 fontstruct
character*256  string
integer*4 count
integer*4 width8

width8 = fxtextwidth( fontstruct, string, count)
```

fontstruct Specifies the address of type **XFontStruct** that has the font information used for the width computation.

string Specifies the character string of 8-bit characters.

count Specifies the character count in the named string.

fxtextwidth determines the width of an 8-bit character string.

The return value *width8* is the width of an 8-bit character string.

For more information about the C language **Xlib** routine called by the **FXlib** binding routine **fxtextwidth**, see the **XTextWidth** routine in "Xlib Functions."

¦ Copyright IBM Corp. 1985, 1991

3.5.356 fxtextwidth16

integer*4 fxtextwidth16
external fxtextwidth16
integer*4 *fontstruct*
integer*4 *string*
integer*4 *count*
integer*4 *width16*

width16 = **fxtextwidth16**(*fontstruct, string, count*)

fontstruct Specifies the address of type **XFontStruct** that has the font information used for the width computation.

string Specifies the address of type **XChar2b** that has a character string of 16-bit characters.

count Specifies the character count in the named string.

fxtextwidth16 determines the width of a 16-bit character string.

The return value *width16* is the width of a 16-bit character string.

For more information about the C language **Xlib** routine called by the **FXlib** binding routine **fxtextwidth16**, see the **XTextWidth16** routine in "Xlib Functions."

¦ Copyright IBM Corp. 1985, 1991

3.5.357 fxtranslatecoordinates

```
integer*4 fxtranslatecoordinates
external  fxtranslatecoordinates
integer*4 display
integer*4 srcw
integer*4 destw
integer*4 srcx, srcy
integer*4 destxreturn, destyreturn
integer*4 childreturn
integer*4 rc

rc = fxtranslatecoordinates( display, srcw, destw, srcx, srcy,
*                                    destxreturn, destyreturn, childreturn)
```

display	Specifies the connection to the X Server.
srcw	Specifies window ID of the source window.
destw	Specifies window ID of the destination window.
srcx, srcy	Specify the *x* and *y* coordinates within the source window.
destxreturn, destyreturn	Return the *x* and *y* coordinates within the destination window.
childreturn	Returns the child if the coordinates are contained in a mapped child of the destination window.

fxtranslatecoordinates transforms a coordinate from the coordinate space of one window to another window or determines which subwindow a coordinate lies in and avoids any race conditions in the process.

For more information about the C language **Xlib** routine called by the **FXlib** binding routine **fxtranslatecoordinates**, see the **XTranslateCoordinates** routine in "Xlib Functions."

¦ Copyright IBM Corp. 1985, 1991

3.5.358 fxundefinecursor

external fxundefinecursor
integer*4 *display*
integer*4 *window*

call fxundefinecursor(*display, window*)

display Specifies the connection to the X Server.

window Specifies the window ID.

fxundefinecursor undefines the mouse or pointer cursor in the specified window.

This routine can generate the error **BadWindow**.

For more information about the C language **Xlib** routine called by the **FXlib** binding routine **fxundefinecursor**, see the **XUndefineCursor** routine in "Xlib Functions."

¦ Copyright IBM Corp. 1985, 1991

3.5.359 fxungrabbutton

```
external fxungrabbutton
integer*4 display
integer*4 buttonungrab
integer*4 modifiers, ungrabwindow

call fxungrabbutton( display, buttonungrab, modifiers,
*                          ungrabwindow)
```

display Specifies the connection to the X Server.

buttonungrab Specifies the pointer button to be released in combination with the modifier bits. This variable can be set to **AnyButton**.

modifiers Specifies the set of keymasks. This mask is a bitwise inclusive OR of valid keymask bits.

ungrabwindow Specifies the window ID of the window to be ungrabbed.

fxungrabbutton ungrabs a mouse or pointer button.

The variable *modifiers* can include the following valid keymask bits:

ShiftMask	**LockMask**
ControlMask	**ModMask**
Mod2Mask	**Mod3Mask**
Mod4Mask	**Mod5Mask**

This variable can also be set to **AnyModifier**, which is equivalent to issuing the grab request for all possible modifier combinations, including the combination of no modifiers.

This routine can generate the error **BadWindow**.

For more information about the C language **Xlib** routine called by the **FXlib** binding routine **fxungrabbutton**, see the **XUngrabButton** routine in "Xlib Functions."

¦ Copyright IBM Corp. 1985, 1991

3.5.360 fxungrabkey

external fxungrabkey
integer*4 *display*
integer*4 *keycode, modifiers, ungrabwindow*

call fxungrabkey(*display, keycode, modifiers, ungrabwindow*)

display Specifies the connection to the X Server.

keycode Specifies the keycode that maps to the specific key to be grabbed. This variable can be the keycode or **AnyKey**.

modifiers Specifies the set of keymasks. This mask is a bitwise inclusive OR of valid keymask bits.

ungrabwindow Specifies the window ID of the window associated with the keyboard to be ungrabbed.

fxungrabkey ungrabs a key.

The variable *modifiers* can include the following valid keymask bits:

ShiftMask **LockMask**

ControlMask **ModMask**

Mod2Mask **Mod3Mask**

Mod4Mask **Mod5Mask**

This variable can also be **AnyModifier**, which is equivalent to issuing the grab request for all possible modifier combinations, including the combination of no modifiers.

fxungrabkey can generate the error **BadWindow**.

For more information about the C language **Xlib** routine called by the **FXlib** binding routine **fxungrabkey**, see the **XUngrabKey** routine in "Xlib Functions."

¦ Copyright IBM Corp. 1985, 1991

3.5.361 fxungrabkeyboard

external fxungrabkeyboard
integer*4 *display*
integer*4 *time*

call fxungrabkeyboard(*display, time*)

display Specifies the connection to the X Server.

time Specifies the time in a time stamp in milliseconds or **CurrentTime**.

fxungrabkeyboard ungrabs the keyboard.

For more information about the C language **Xlib** routine called by the **FXlib** binding routine **fxungrabkeyboard**, see the **XUngrabKeyboard** routine in "Xlib Functions."

¦ Copyright IBM Corp. 1985, 1991

3.5.362 fxungrabpointer

external fxungrabpointer
integer*4 *display*
integer*4 *time*

call fxungrabpointer(*display*, *time*)

display Specifies the connection to the X Server.

time Specifies the time in a time stamp in milliseconds or **CurrentTime**.

fxungrabpointer ungrabs the pointer.

For more information about the C language **Xlib** routine called by the **FXlib** binding routine **fxungrabpointer**, see the **XUngrabPointer** routine in "Xlib Functions."

¦ Copyright IBM Corp. 1985, 1991

3.5.363 fxungrabserver

external fxungrabserver
integer*4 *display*

call fxungrabserver(*display*)

display Specifies the connection to the X Server.

fxungrabserver ungrabs the server.

For more information about the C language **Xlib** routine called by the **FXlib** binding routine **fxungrabserver**, see the **XUngrabServer** routine in "Xlib Functions."

¦ Copyright IBM Corp. 1985, 1991

3.5.364 fxuninstallcolormap

```
external fxuninstallcolormap
integer*4 display
integer*4 cmap

call fxuninstallcolormap( display, cmap)
```

display Specifies the connection to the X Server.

cmap Specifies the colormap ID.

fxuninstallcolormap uninstalls a colormap.

This routine can generate the error **BadColor**.

For more information about the C language **Xlib** routine called by the **FXlib** binding routine **fxuninstallcolormap**, see the **XUninstallColormap** routine in "Xlib Functions."

¦ Copyright IBM Corp. 1985, 1991

3.5.365 fxunionrectwithregion

external fxunionrectwithregion
integer*4 *rectangle*
integer*4 *srcregion*
integer*4 *destregion*

call fxunionrectwithregion(*rectangle, srcregion, destreg ion*1.)

rectangle Specifies the rectangle.

srcregion Specifies the source region to be used.

destregion Specifies the destination region.

fxunionrectwithregion computes the union of two regions.

For more information about the C language **Xlib** routine called by the **FXlib** binding routine **fxunionrectwithregion**, see the **XUnionRectWithRegion** routine in "Xlib Functions."

¦ Copyright IBM Corp. 1985, 1991

3.5.366 fxunionregion

external fxunionregion
integer*4 *sra, srb, dr*

call fxunionregion(*sra, srb, dr*)

sra, srb Specify the addresses of type **_XRegion** of the two region structures with which to perform the computation.

dr Specifies the address of type **_XRegion** of the resulting region structure.

fxunionregion computes the union of two regions.

For more information about the C language **Xlib** routine called by the **FXlib** binding routine **fxunionregion**, see the **XUnionRegion** routine in "Xlib Functions."

¦ Copyright IBM Corp. 1985, 1991

3.5.367 fxunloadfont

external fxunloadfont
integer*4 *display*
integer*4 *font*

call fxunloadfont(*display, font*)

display Specifies the connection to the X Server.

font Specifies the font ID.

fxunloadfont unloads the font loaded by **fxloadfont**.

This routine can generate the error **BadFont**.

For more information about the C language **Xlib** routine called by the **FXlib** binding routine **fxunloadfont**, see the **XUnloadFont** routine in "Xlib Functions."

¦ Copyright IBM Corp. 1985, 1991

3.5.368 fxunmapsubwindows

```
external  fxunmapsubwindows
integer*4 display
integer*4 window

call fxunmapsubwindows( display, window)
```

display Specifies the connection to the X Server.

window Specifies the window ID.

fxunmapsubwindows unmaps all subwindows for a specified window.

For more information about the C language **Xlib** routine called by the **FXlib** binding routine **fxunmapsubwindows**, see the **XUnmapSubwindows** routine in "Xlib Functions."

¦ Copyright IBM Corp. 1985, 1991

3.5.369 fxunmapwindow

```
external  fxunmapwindow
integer*4 display
integer*4 window

call fxunmapwindow( display, window)
```

display Specifies the connection to the X Server.

window Specifies the window ID.

fxunmapwindow unmaps the specified window.

For more information about the C language **Xlib** routine called by the **FXlib** binding routine **fxunmapwindow**, see the **XUnmapWindow** routine in "Xlib Functions."

¦ Copyright IBM Corp. 1985, 1991

3.5.370 fxusekeymap

external fxusekeymap
integer*4 fxusekeymap
integer*4 *status*
integer*4 *keymapfile*

status = fxusekeymap(*keymapfile*)

keymapfile Specifies the name of the keymap file to use within the current process.

fxusekeymap changes the keymap file used for the current process.

The return value status contains a zero on return if it cannot find the keymap file named by *keymapfile* or if the file contains an invalid magic number.

For more information about the C language **Xlib** routine called bythe **FXlib** binding routine **fxusekeymap**, see the **XUseKeymap** routine in "Xlib Functions" in topic 2.4.

¦ Copyright IBM Corp. 1985, 1991

3.5.371 fxvisualidfromvisual

```
external fxvisualidfromvisual
integer*4 visual
integer*4 display

visual = fxvisualidfromvisual( display)
```

display Specifies the connection to the X Server.

visual Specifies the visual type

fxvisualidfromvisual returns the visual ID for the specified visual type.

For more information about the C language **Xlib** routine called bythe **FXlib** binding routine **fxvisualidfromvisual**, see the **XVisualIDFromVisual** routine in "Xlib Functions" in topic 2.4.

¦ Copyright IBM Corp. 1985, 1991

3.5.372 fxvendorrelease

```
integer*4  fxvendorrelease
external   fxvendorrelease
integer*4  display
integer*4  rc

rc = fxvendorrelease( display)
```

display Specifies the connection to the X Server.

fxvendorrelease obtains a number related to a vendor's release of the X Server.

Return value *rc* is the value of **fxvendorrelease**.

For more information about the C language **Xlib** routine called by the **FXlib** binding routine **fxvendorrelease**, see the **XVendorRelease** routine in "Xlib Functions."

¦ Copyright IBM Corp. 1985, 1991

3.5.373 fxwarppointer

```
external fxwarppointer
integer*4 display
integer*4 srcw, destw
integer*4 srcx, srcy
integer*4 srcwidth, srcheight
integer*4 destx, desty

call fxwarppointer( display, srcw, destw, srcx, srcy, srcwidth,
*                       srcheight, destx, desty)
```

display	Specifies the connection to the X Server.
scrw	Specifies the window ID of the source window. This can be the window ID or **None**.
destw	Specifies the window ID of the destination window. This can be the window ID or **None**.
srcx, srcy	Specify the *x* and *y* coordinates within the source window.
srcwidth, srcheight	Specify the *width* and *height* of the source window.
destx, desty	Specify the *x* and *y* coordinates within the destination window.

fxwarppointer moves the pointer to an arbitrary point on the screen.

This routine can generate the error **BadWindow**.

For more information about the C language **Xlib** routine called by the **FXlib** binding routine **fxwarppointer**, see the **XWarpPointer** routine in "Xlib Functions."

¦ Copyright IBM Corp. 1985, 1991

3.5.374 fxwhitepixel

```
integer*4  fxwhitepixel
external   fxwhitepixel
integer*4  display
integer*4  screen
integer*4  rc

rc = fxwhitepixel( display, screen)
```

display Specifies the connection to the X Server.

screen Specifies the screen.

fxwhitepixel obtains the white pixel for the screen specified.

Return value *rc* is the value of the white pixel.

For more information about the C language **Xlib** routine called by the **FXlib** binding routine **fxwhitepixel**, see the **XWhitePixel** routine in "Xlib Functions."

¦ Copyright IBM Corp. 1985, 1991

3.5.375 fxwhitepixelofscreen

```
integer*4  fxwhitepixelofscreen
external   fxwhitepixelofscreen
integer*4  screen
integer*4  rc

rc = fxwhitepixelofscreen( screen)
```

screen Specifies the address of type **Screen** of the screen of the display.

fxwhitepixelofscreen returns the white pixel value of the screen specified.

Return value *rc* is the white pixel of the screen specified.

For more information about the C language **Xlib** routine called by the **FXlib** binding routine **fxwhitepixelofscreen**, see the **XWhitePixelOfScreen** routine in "Xlib Functions."

¦ Copyright IBM Corp. 1985, 1991

3.5.376 fxwidthmmofscreen

```
integer*4  fxwidthmmofscreen
external   fxwidthmmofscreen
integer*4  screen
integer*4  rc

rc = fxwidthmmofscreen( screen)
```

screen Specifies the address of type **Screen** of the screen of the display.

fxwidthmmofscreen returns the width in millimeters of the screen specified.

Return value *rc* is the width of the screen in millimeters.

For more information about the C language **Xlib** routine called by the **FXlib** binding routine **fxwidthmmofscreen**, see the **XWidthMMOfScreen** routine in "Xlib Functions."

¦ Copyright IBM Corp. 1985, 1991

3.5.377 fxwidthofscreen

```
integer*4  fxwidthofscreen
external   fxwidthofscreen
integer*4  screen
integer*4  rc

rc = fxwidthofscreen( screen)
```

screen Specifies the address of type **Screen** of the screen of the display.

fxwidthofscreen returns the width of the screen specified.

Return value *rc* is the width of the screen in pixels.

For more information about the C language **Xlib** routine called by the **FXlib** binding routine **fxwidthofscreen**, see the **XWidthOfScreen** routine in "Xlib Functions."

¦ Copyright IBM Corp. 1985, 1991

3.5.378 fxwindowevent

external fxwindowevent
integer*4 *display*
integer*4 *window, eventmask, eventreturn*

call fxwindowevent(*display, window, eventmask, eventreturn*)

display Specifies the connection to the X Server.

window Specifies the window ID.

eventmask Specifies the eventmask. This mask is a bitwise inclusive or of valid event mask bits.

eventreturn Specifies the destination address of type **XEvent** in the client data area to copy the event structure from the event queue. The client must allocate the data area. The client must also free the data area when it is no longer needed.

fxwindowevent removes the next event that matches both a window and an event mask.

For more information about the C language **Xlib** routine called by the **FXlib** binding routine **fxwindowevent**, see the **XWindowEvent** routine in "Xlib Functions."

¦ Copyright IBM Corp. 1985, 1991

3.5.379 fxwritebitmapfile

```
integer*4 fxwritebitmapfile
external  fxwritebitmapfile
integer*4 display
character*256  filename
integer*4 bitmap, width, height, xhot, yhot
integer*4 rc

rc = fxwritebitmapfile( display, filename, bitmap, width,
*                              height, xhot, yhot)
```

display Specifies the connection to the X Server.

filename Specifies the filename to use. The format of the filename is specific to the operating system.

bitmap Specifies the bitmap to be written.

width, height Specify the *width* and *height* dimensions of the bitmap to be written.

xhot, yhot Specify where in the file to place the hot spot coordinates. If none is present, it specifies (-1,-1).

fxwritebitmapfile writes a bitmap onto a file.

Return code *rc* returns **BitmapOpenFailed** if the file cannot be opened for writing, **BitmapNoMemeory** if not enough memory is allocated, or **BitmapSuccess** if no error is encountered.

For more information about the C language **Xlib** routine called by the **FXlib** binding routine **fxwritebitmapfile**, see the **XWriteBitmapFile** routine in "Xlib Functions."

¦ Copyright IBM Corp. 1985, 1991

3.5.380 fxxorregion

external fxxorregion
integer*4 *sra, srb, dr*

call fxxorregion(*sra, srb, dr*)

sra, srb Specify the addresses of type **_XRegion** of the two region structures with which to perform the computation.

dr Specifies the address of type **_XRegion** of the resulting region structure.

fxxorregion calculates the difference between the union and intersection of two regions.

For more information about the C language **Xlib** routine called by the **FXlib** binding routine **fxxorregion**, see the **XXorRegion** routine in "Xlib Functions."

¦ Copyright IBM Corp. 1985, 1991

4.0 Chapter 4. Toolkit

Subtopics

¦ Copyright IBM Corp. 1985, 1991

4.1 CONTENTS

¦ Copyright IBM Corp. 1985, 1991

4.2 About This Chapter

The X-Windows Toolkit provides basic functions for building a wide variety of application environments and gives programmers a common set of underlying user-interface functions. These tools simplify the design of application user interfaces in the X-Windows programming environment.

This chapter explains how the Toolkit works and contains the procedure types and routines (in alphabetical sequence) with command syntax and definition of variables.

¦ Copyright IBM Corp. 1985, 1991

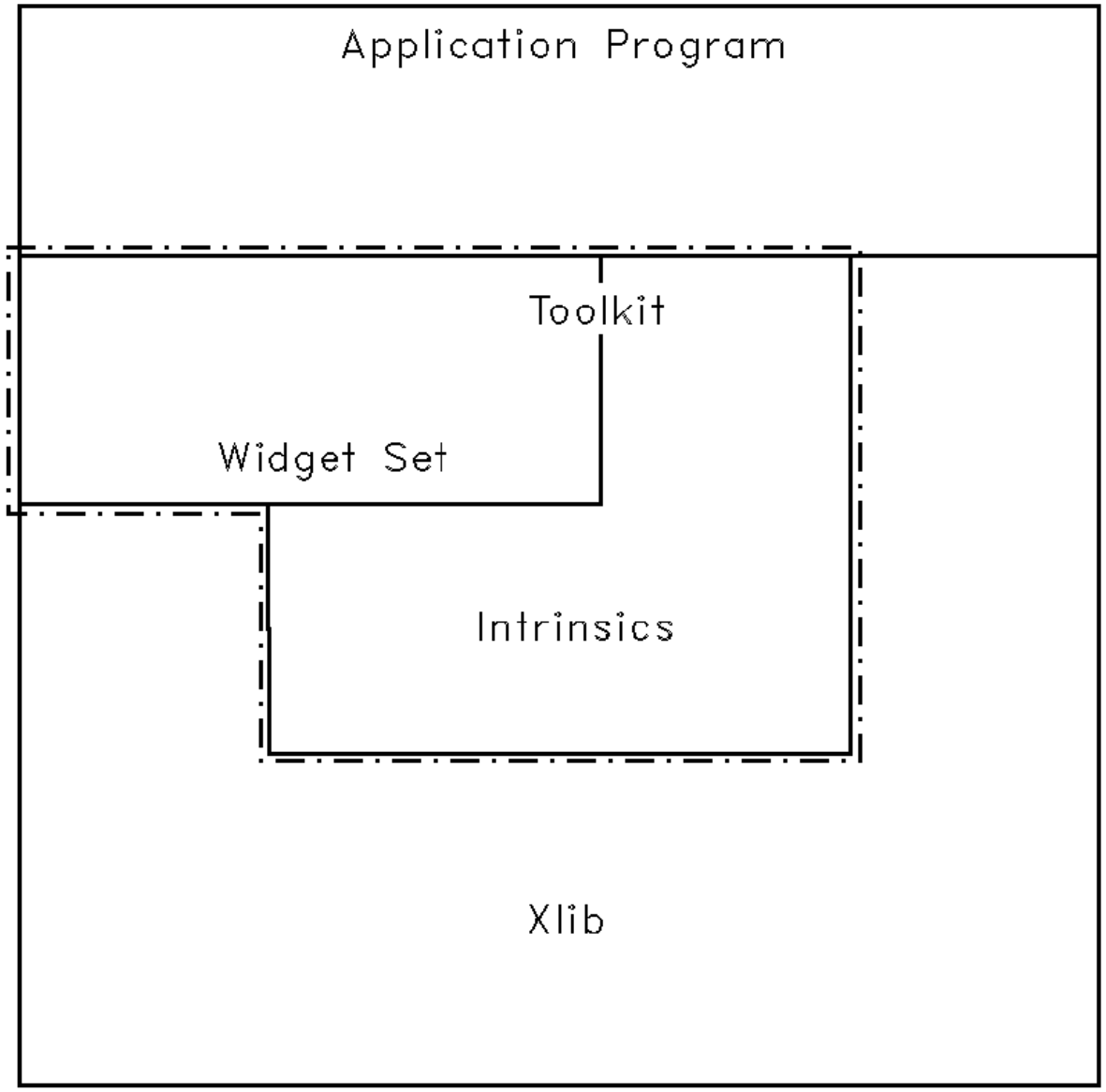
4.3 Intrinsics and Widgets

The Intrinsics and a widget set make up the X Toolkit. The Intrinsics provide the base mechanisms necessary to build a wide variety of widget sets and application environments. Because the Intrinsics mask implementation details from the widget and the application programmer, the widgets and the application environments built with these widgets are fully extensible and support user-developed or extended components. By following a small set of conventions, widget programmers can extend their widget sets in new ways and can have these extensions function with the existing facilities.

The Intrinsics is a library package layered on **Xlib**. As such, the Intrinsics provide functions and structures for extending the basic programming abstractions of X-Windows. By providing these funcions and structures for intercomponent and intracomponent interactions, the Intrinsics provide the next layer of functionality from which the widget sets are built.

This X programming environment can be illustrated as three tiers, all sitting beneath the application program. The basic support (bottom tier) is the **C** language X-Windows routines (**Xlib**). The Intrinsics is on the **Xlib**. The widget set is on the Intrinsics. The combination of the Intrinsics and the widget set compose the Toolkit.

¦ Copyright IBM Corp. 1985, 1991



The X-Windows toolkit application is usually a client of a given widget set, a subset of the Intrinsics functions, and a smaller set of **Xlib** functions. At the same time, a widget set is a client of both the Intrinsics and **Xlib**. And, the Intrinsics are a client of **Xlib** only.

For the application programmer, the X-Windows Toolkit provides the following:

A consistent interface (widget set) for writing application

A set of Intrinsics mechanisms (functions and structures) used als for writing applications.

For the widget programmer, the X-Windows Toolkit provides:

A set of Intrinsics mechanisms for building widgets

An architectural model for constructing and composing widgets

Subtopics

¦ Copyright IBM Corp. 1985, 1991

4.3.1 Requirements

¦ Copyright IBM Corp. 1985, 1991

4.3.1 Requirements

Applications that use the Toolkit must include the following header files:

```
<X11/Xlib.h>
<X11/Intrinsic.h>
<X11/StringDefs.h>
```

and possibly **<X11/Shell.h>**.

Widget implementations should include **<X11/IntrinsicP.h>** instead of **<X11/Intrinsic.h>**.

The applications should also include the additional headers for each widget class to be used, such as **<X11/Label.h>** or **<X11/Scroll.h>**. The Intrinsics object library file is named **libXt.a**.

¦ Copyright IBM Corp. 1985, 1991

4.4 Defining Widgets

The fundamental data type of the X-Windows Toolkit is the ***widget***, a combination of a window and its associated semantics. A widget is dynamically allocated and contains state information. Every widget belongs to one ***widget class*** that is allocated statically and initialized. The widget class contains the operations allowed on widgets of that class.

Logically, a widget is a rectangle with associated input and output semantics. Some widgets display information, such as text or graphics, while others are containers for other widgets, such as a menu box. Some widgets are output only and do not react to pointer or keyboard input, while others change their display in response to input and can invoke functions attached to them by an application. The user can alter much of the input and output of a widget, such as fonts, colors, sizes, and border widths.

A ***widget instance*** is composed of two parts:

A data structure that contains instance-specific value
A class structure that contains information that is applicable to al widgets of that class.

Each widget class is logically composed of the procedures and data that is associated with all widgets belonging to that class. These procedures and data can be inherited by subclasses.

Physically, a widget class is a pointer to a structure. The contents of this structure are constant for all widgets of the widget class, even though the values can vary from widget class to widget class. (Here, ***constant*** means the class structure is initialized at compile-time and never changed, except for a one-time class initialization and in-place compilation of resource lists. Compilation occurs when the first widget of the class or subclass is created.) A widget instance is allocated and initialized by **XtCreateWidget**. See "Creating Widgets" in topic 4.7.

The organization of the declarations and code for a new widget class between a public **.h** file, a private **.h** file, and the implementation **.c** file is described in "Defining Widget Classes" in topic 4.5. The predefined widget classes adhere to these conventions.

These predefined widget classes include:

Core widget class	Defines the fields common to all widgets. All widget classes are subclasses of Core, which is defined by the **CoreClassPart** and **CorePart** structures.
Composite widget class	Exists as a subclass of the Core class and defines widgets that act as containers of other widgets. Composite widgets are defined by the **CompositeClassPart** and **CompositePart** structures.
Constraint widget class	Exists as a subclass of the Composite class and defines widgets that maintain additional state data for each child. This data may include client-defined constraints on the child's geometry. Constraint widgets are defined by the **ConstraintClassPart** and **ConstraintPart** structures.

¦ Copyright IBM Corp. 1985, 1991

Subtopics
4.4.1 Naming Widgets
4.4.2 Core Widget Class
4.4.3 Composite widget class
4.4.4 Constraint Widget Class

¦ Copyright IBM Corp. 1985, 1991

4.4.1 Naming Widgets

The Intrinsics provide the ability to create new widgets and organize a set of widgets into an application. Use the following guidelines for naming new widgets:

Use applicable X-Windows naming conventions

- A *record component* name is written in lower-case letters. If a name contains more than one word, the words are connected by underscores (_). For example, *background_pixmap* is a compound record component name.
- The first letter of *procedure* and *type* names is capitalized. Compound type and procedure names are capitalized at the beginning of each part of the name. For example, **XtArgList** is a *compound procedure* name and **XtInputCallbackProc** is a *compound type* name.

A *resource* name string is spelled the same as the *record component* name, except that capitalization is used for compound names. For the compiler to find spelling errors, each resource name should have a macro definition prefixed with **XtN**. For example, the *background_pixmap* field has the corresponding resource name identifier **XtNbackgroundPixmap**, which is defined as the string **backgroundPixmap**. Since many predefined names are listed in the **<X11/StringDefs.h>** file, make sure any new name you create is not in this file.

A *resource class* string starts with a capital letter and uses capitalization in compound names. For example, "BorderWidth" is a resource class compound name. Each resource class string should have a macro definition prefixed with **XtC**, for example, **XtCBorderWidth**.

A *resource representation* string is spelled the same as the type name, such as **TranslationTable**. Each representation string should have a macro definition prefixed with **XtR**, for example, **XtRTranslationTable**.

New widget class names begin with a capital letter. Capitalization i used to form compound words. For example, the following names can be derived from the new class name **AbcXyz**:

Example	**Name of**
AbcXyzPart	A partial widget instance structure
AbcXyzRec and **_AbcXyzRec**	A complete widget instance structure
AbcXyzWidget	A widget instance pointer type
AbcXyzClassPart	A partial class structure
AbcXyzClassRec and **_AbcXyzClassRec**	A complete class structure
abcXyzClassRec	A class structure variable
abcXyzWidgetClass	A class pointer variable.

Action procedures available to translation specifications shoul follow the same naming conventions as procedures. Capitalize the first letter and use capitalization for compound names, for example, **Highlight** and **NotifyClient**.

¦ Copyright IBM Corp. 1985, 1991

4.4.2 Core Widget Class

The Core widget class defines the fields common to all widgets. All widget classes are subclasses of Core, which is defined by the **CoreClassPart** and **CorePart** structures.

Subtopics
4.4.2.1 Defining the CoreClassPart Structure
4.4.2.2 Defining the CorePart Structure
4.4.2.3 Default Values for the CorePart Structure

¦ Copyright IBM Corp. 1985, 1991

4.4.2.1 Defining the CoreClassPart Structure

The common fields for all widget classes are defined in the **CoreClassPart** structure:

typedef struct {	**See ...**
WidgetClass *superclass*;	"Defining Widget Classes" in topic 4.5.
String *class_name*;	"Defining Widget Classes" in topic 4.5.
Cardinal *widget_size*;	"Creating Widgets" in topic 4.7.
XtProc *class_initialize*;	"Defining Widget Classes" in topic 4.5.
XtWidgetClassProc *class_part_initialize*;	"Defining Widget Classes" in topic 4.5.
Boolean *class_inited*;	"XtCreateWidget" in topic 4.17.69. (Private field)
XtInitProc *initialize*;	"Creating Widgets" in topic 4.7.
XtArgsProc *initialize_hook*;	"Creating Widgets" in topic 4.7.
XtRealizeProc *realize*;	"Creating Widgets" in topic 4.7.
XtActionList *actions*;	"Using Translation Management" in topic 4.16.
Cardinal *num_actions*;	"Using Translation Management" in topic 4.16.
XtResourceList *resources*;	"Resource Management" in topic 4.14.
Cardinal *num_resources*;	"Resource Management" in topic 4.14.
XrmClass *xrm_class*;	"Resource Management" in topic 4.14 (Private field)
Boolean *compress_motion*;	"Compressing Events Using Event Filters" in topic 4.12.2.
Boolean *compress_exposure*;	"Compressing Events Using Event Filters" in topic 4.12.2.
Boolean *compress_enterleave*;	"Compressing Events Using Event Filters" in

¦ Copyright IBM Corp. 1985, 1991

	topic 4.12.2.
Boolean *visible_interest*;	"Handling Widget Exposure and Visibility" in topic 4.12.10.
XtWidgetProc *destroy*;	"Destroying Widgets" in topic 4.7.7.
XtWidgetProc *resize*;	"Managing Widget Geometry" in topic 4.13.
XtExposeProc *expose*;	"Handling Widget Exposure and Visibility" in topic 4.12.10.
XtSetValuesFunc *set_values*;	"Reading and Writing Widget State" in topic 4.15.3.1.
XtArgsFunc *set_values_hook*;	"Reading and Writing Widget State" in topic 4.15.3.1.
XtAlmostProc *set_values_almost*;	"Reading and Writing Widget State" in topic 4.15.3.1.
XtArgsProc *get_values_hook*;	"Reading and Writing Widget State" in topic 4.15.3.1.
XtAcceptFocusProc *accept_focus*;	"XtSetKeyboardFocus" in topic 4.17.163.
XtVersionType *version*;	"Defining Widget Classes" in topic 4.5.
_XtOffsetList *callback_private*;	"Using Callbacks" in topic 4.7.8. (Private field)
String *tm_table*;	"Using Translation Management" in topic 4.16.
XtGeometryHandler *query_geometry*;	"Managing Widget Geometry" in topic 4.13.
XtStringProc *display_accelerator*;	"Using Accelerators" in topic 4.16.6.
caddr_t *extension*;	"Defining Widget Classes" in topic 4.5.

} CoreClassPart;

All widget classes have the Core class fields as their first component. The prototypical type **WidgetClass** is defined with the Core class fields only. Various routines can cast widget class pointers, as needed, to specific widget class types. For example:

```
typedef struct {
     CoreClassPart core_class;
} WidgetClassRec, *WidgetClass;
```

¦ Copyright IBM Corp. 1985, 1991

The predefined class record and pointer for **WidgetClassRec** are:

```
extern WidgetClassRec widgetClassRec;
extern WidgetClass widgetClass;
```

The opaque types, **Widget** and **WidgetClass**, and the opaque variable, *widgetClass*, are defined for generic actions on widgets.

¦ Copyright IBM Corp. 1985, 1991

4.4.2.2 Defining the CorePart Structure

The common fields for all widget instances are defined in the **CorePart** structure:

typedef struct _CorePart{	**See ...**
Widget *self*;	"Defining Widgets" in topic 4.4.
WidgetClass *widget_class*;	"Defining Widget Classes" in topic 4.5.
Widget *parent*;	"Defining Widget Classes" in topic 4.5.
XrmName *xrm_name*;	"Resource Management" in topic 4.14. (Private field)
Boolean *being_destroyed*;	"Destroying Widgets" in topic 4.7.7.
XtCallbackList *destroy_callbacks*;	"Destroying Widgets" in topic 4.7.7.
caddr_t *constraints*;	"Using Constrained Composite Widgets" in topic 4.8.6.
Position *x*;	"Managing Widget Geometry" in topic 4.13.
Position *y*;	"Managing Widget Geometry" in topic 4.13.
Dimension *width*;	"Managing Widget Geometry" in topic 4.13.
Dimension *height*;	"Managing Widget Geometry" in topic 4.13.
Dimension *border_width*;	"Managing Widget Geometry" in topic 4.13.
Boolean *managed*;	"Using Composite Widgets" in topic 4.8.
Boolean *sensitive*;	"XtSetSensitive" in topic 4.17.167 and "XtIsSensitive" in topic 4.17.108.
Boolean *ancestor_sensitive*;	"XtSetSensitive" in topic 4.17.167 and "XtIsSensitive" in topic 4.17.108.
XtEventTable *event_table*;	"Managing Events" in topic 4.12. (Private field)
XtTMRec *tm*;	"Using Translation Management" in topic 4.16. (Private field)
XtTranslations *accelerators*;	"Using Accelerators" in topic 4.16.6.

¦ Copyright IBM Corp. 1985, 1991

Pixel *border_pixel*;	"Obtaining Window Information" in topic 4.7.6.
Pixmap *border_pixmap*;	"Obtaining Window Information" in topic 4.7.6.
WidgetList *popup_list*;	"Using Pop-up Widgets" in topic 4.10.
Cardinal *num_popups*;	"Using Pop-up Widgets" in topic 4.10.
String *name*;	"Resource Management" in topic 4.14.
Screen **screen*;	"Obtaining Window Information" in topic 4.7.6.
Colormap *colormap*;	"Obtaining Window Information" in topic 4.7.6.
Window *window*;	"Obtaining Window Information" in topic 4.7.6.
Cardinal *depth*;	"Obtaining Window Information" in topic 4.7.6.
Pixel *background_pixel*;	"Obtaining Window Information" in topic 4.7.6.
Pixmap *background_pixmap*;	"Obtaining Window Information" in topic 4.7.6.
Boolean *visible*;	"Handling Widget Exposure and Visibility" in topic 4.12.10.
Boolean *mapped_when_managed*;	"Using Composite Widgets" in topic 4.8.

} **CorePart**;

All widget instances have the core fields as their first component. The prototypical type **Widget** is defined with the core set of fields only. Various routines can cast widget pointers, as needed, to specific widget types. For example:

```
typedef struct {
     CorePart core;
} WidgetRec, *Widget;
```

¦ Copyright IBM Corp. 1985, 1991

4.4.2.3 Default Values for the CorePart Structure

The default values for the core fields are filled in by the Core resource list and the Core initialize procedure. The default values for the **CorePart** structure are:

Field	**Default Value**
self	Address of the widget structure (may not be changed)
widget_class	*widget_class* argument to **XtCreateWidget** (may not be changed)
parent	*parent* argument to **XtCreateWidget** (may not be changed)
xrm_name	Encoded *name* argument to **XtCreateWidget** (may not be changed)
being_destroyed	*being_destroyed* value of the parent widget
destroy_callbacks	**NULL**
constraints	**NULL**
x	0
y	0
width	0
height	0
border_width	1
managed	**False**
sensitive	**True**
ancestor_sensitive	Bitwise AND of *sensitive* & *ancestor_sensitive* of the parent widget
event_table	Initialized by the event manager
tm	Initialized by the translation manager
accelerators	**NULL**
border_pixel	**XtDefaultForeground**
border_pixmap	**NULL**
popup_list	**NULL**
num_popups	0
name	*name* argument to **XtCreateWidget** (may not be changed)
screen	Parent screen; top-level widget uses display specifier (may not be changed)
colormap	Default colormap for the screen
window	**NULL**
depth	Parent's depth; top-level widget uses root window depth
background_pixel	**XtDefaultBackground**
background_pixmap	**NULL**
visible	**True**
map_when_managed	**True**.

¦ Copyright IBM Corp. 1985, 1991

4.4.3 Composite widget class

The Composite widget class exists as a subclass of the Core class and defines widgets that act as containers of other widgets. Composite widgets are defined by the **CompositeClassPart** and **CompositePart** structures.

Subtopics
4.4.3.1 Defining CompositeClassPart Structure
4.4.3.2 Defining the CompositePart Structure
4.4.3.3 Default Values for the CompositePart Structure

¦ Copyright IBM Corp. 1985, 1991

4.4.3.1 Defining CompositeClassPart Structure

In addition to the Core widget class fields, Composite widgets have the following class fields:

typedef struct {	**See ...**
XtGeometryHandler *geometry_manager*;	"Managing Widget Geometry" in topic 4.13.
XtWidgetProc *change_managed*;	"Using Composite Widgets" in topic 4.8.
XtWidgetProc *insert_child*;	"Using Composite Widgets" in topic 4.8.
XtWidgetProc *delete_child*;	"Using Composite Widgets" in topic 4.8.
caddr_t *extension*;	"Defining Widget Classes" in topic 4.5.

} CompositeClassPart;

Composite widget classes have the composite fields immediately after the core fields:

```
typedef struct {
     CoreClassPart core_class;
     CompositeClassPart composite_class;
}  CompositeClassRec, *CompositeWidgetClass;
```

The predefined class record and pointer for **CompositeClassRec** are:

```
extern CompositeClassRec compositeClassRec;
extern WidgetClass compositeWidgetClass;
```

The opaque types, **CompositeWidget** and **CompositeWidgetClass**, and the opaque variable, *compositeWidgetClass*, are defined for generic operations on widgets that are a subclass of **CompositeWidget**.

¦ Copyright IBM Corp. 1985, 1991

4.4.3.2 Defining the CompositePart Structure

In addition to the CorePart fields, composite widgets have the following fields defined in the **CompositePart** structure:

typedef struct {	**See ...**
WidgetList *children*;	"Defining Widget Classes" in topic 4.5.
Cardinal *num_children*;	"Defining Widget Classes" in topic 4.5.
Cardinal *num_slots*;	"Using Composite Widgets" in topic 4.8.
XtOrderProc *insert_position*;	"Creating Widgets" in topic 4.7.
} CompositePart;	

Composite widgets have the composite fields immediately after the core fields:

```
typedef struct {
        CorePart core;
        CompositePart composite;
}  CompositeRec, *CompositeWidget;
```

¦ Copyright IBM Corp. 1985, 1991

4.4.3.3 Default Values for the CompositePart Structure

The default values are filled in by the Composite resource list and the Composite initialize procedure. The default values for the **CompositePart** structure are:

Field	**Default Value**
children	**NULL**
num_children	0
num_slots	0
insert_position	Internal function **InsertAtEnd**.

¦ Copyright IBM Corp. 1985, 1991

4.4.4 Constraint Widget Class

The Constraint widget class exists as a subclass of the Composite class and defines widgets that maintain additional state data for each child. This data may include client-defined contraints on the geometry of the child. Constraint widgets are defined by the **ConstraintClassPart** and **ConstraintPart** structures.

Subtopics
4.4.4.1 Defining ConstraintClassPart Structure
4.4.4.2 Defining ConstraintPart Structure

¦ Copyright IBM Corp. 1985, 1991

4.4.4.1 Defining ConstraintClassPart Structure

In addition to the Composite class fields, Constraint widgets have the following class fields:

typedef struct {	**See ...**
XtResourceList *resources*;	"Using Constrained Composite Widgets" in topic 4.8.6.
Cardinal *num_resources*;	"Using Constrained Composite Widgets" in topic 4.8.6.
Cardinal *constraint_size*;	"Using Constrained Composite Widgets" in topic 4.8.6.
XtInitProc *initialize*;	"Using Constrained Composite Widgets" in topic 4.8.6.
XtWidgetProc *destroy*;	"Using Constrained Composite Widgets" in topic 4.8.6.
XtSetValuesFunc *set_values*;	"Using Constrained Composite Widgets" in topic 4.8.6.
caddr_t *extension*;	"Defining Widget Classes" in topic 4.5.
} ConstraintClassPart;	

Constraint widget classes have the constraint fields immediately after the composite fields:

```
typedef struct {
     CoreClassPart core_class;
     CompositeClassPart composite_class;
     ConstraintClassPart constraint_class;
} ConstraintClassRec, *ConstraintWidgetClass;
```

The predefined class record and pointer for **ConstraintClassRec** are:

```
extern ConstraintClassRec constraintClassRec;
extern WidgetClass constraintWidgetClass;
```

The opaque types **ConstraintWidget** and **ConstraintWidgetClass**, and the opaque variable, *constraintWidgetClass*, are defined for generic operations on widgets that are a subclass of **ConstraintWidgetClass**.

¦ Copyright IBM Corp. 1985, 1991

4.4.4.2 Defining ConstraintPart Structure

In addition to the **CompositePart** fields, Constraint widgets have the following fields defined in the **ConstraintPart** structure:

```
typedef struct {  int empty; } ConstraintPart;
```

Constraint widgets have the *constraint* fields immediately after the composite fields:

```
typedef struct {
     CorePart core;
     CompositePart composite;
     ConstraintPart constraint;
}  ConstraintRec, *ConstraintWidget;
```

¦ Copyright IBM Corp. 1985, 1991

4.5 Defining Widget Classes

The ***widget_class*** field of a widget points to its widget class structure that contains information that is constant for all widgets of that class.

With this class-oriented structure, widget classes do not usually implement procedures directly. Rather, widgets implement procedures available through their widget class structure. These class procedures are invoked by generic procedures that envelop common actions around the procedures implemented by the widget class. Such procedures are applicable to all widgets of that class and to widgets that are subclasses of that class.

All widget classes are a subclass of the Core class and can be subclassed further. Subclassing reduces the amount of code and declarations you write to make a new widget class that is similar to an existing class.

For example, you do not have to describe every resource your widget uses in an **XtResourceList**. Instead, describe the resources your widget has that its superclass does not. Subclasses usually inherit many of the procedures of their superclasses, such as the expose procedure or the geometry handler. On the other hand, subclassing too extensively creates a subclass that does not inherit the procedures of its superclass.

To make good use of subclassing, widget declarations and naming conventions are highly stylized. A widget consists of three files:

- A public **.h** file used by client widgets or applications
- A private **.h** file used by widgets that are subclasses of the widget
- A **.c** file that implements the widget class.

Subtopics

¦ Copyright IBM Corp. 1985, 1991

4.5.1 Using Widget Subclassing in Public .h Files

The public **.h** file for a widget class is imported by clients. It contains the following:

A reference to the public **.h** files for the superclass
The names and classes of the new resources to be added to it superclass
The class record pointer used to create widget instance
The C language type, which is used to declare widget instances of thi class
Entry points for new class methods

The following example shows the public **.h** file for a possible implementation of the Label widget:

```
#ifndef LABEL_H
#define LABEL_H

/* New resources */
#define XtNjustify         "justify"
#define XtNforeground      "foreground"
#define XtNlabel           "label"
#define XtNfont            "font"
#define XtNinternalWidth   "internalWidth"
#define XtNinternalHeight  "internalHeight"

/* Class record pointer */
extern WidgetClass labelWidgetClass;

/* C language widget type definition */
typedef struct _LabelRec   *LabelWidget;

/* New class method entry points */
extern void Label SetText();
        /* Widget widget */
        /* String text */

extern String Label GetText();
        /* Widget widget */

#endif LABEL_H
```

Conditionally including the text allows the application to include header files for different widgets without determining whether they have already been included as a superclass of another widget.

To accommodate an operating system with file name length restrictions, the name of the public **.h** file is the first 10 characters of the widget class. For example, the public **.h** file for the Constraint widget is **Constraint.h**.

¦ Copyright IBM Corp. 1985, 1991

4.5.2 Using Widget Subclassing in Private .h Files

The private **.h** file for a widget is imported by widget classes that are subclasses of the widget. It contains:

A reference to the public **.h** file for the class
A reference to the private **.h** file for the superclass
The new fields this widget adds to the widget structure of it superclass
The complete widget instance structure for this widge
The new fields added to the **Constraint** structure of its superclass if the widget is a subclass of **Constraint**
The complete **Constraint** structure if the widget is a subclass of **Constraint**
The new fields added to the widget class structure of its superclas
The complete widget class structure for this widge
The name of the constant for the generic widget class structur
An inherit procedure for subclasses where each new procedure in th widget class structure should inherit a superclass operation.

The following example shows the private **.h** file for the Label widget:

```
#ifndef LABELP_H
#define LABELP_H

#include <X11/Label.h>

/* New fields for the Label widget record */

typedef struct {
/* Settable resources */
   Pixel foreground;
   XFontStruct *font;
   String label;              /* text to display */
   XtJustify justify;
   Dimension internal_width;  /* # of pixels in horizontal border */
   Dimension internal_height; /* # of pixels in vertical border */
/* Data derived from resources */
   GC normal_GC;
   GC gray_GC;
   Pixmap gray_pixmap;
   Position label_x;
   Position label_y;
   Dimension label_width;
   Dimension label_height;
   Cardinal label_len;
   Boolean display_sensitive;
} LabelPart;

/* Full instance record declaration */
typedef struct_LabelRec {
   CorePart core;
   LabelPart label;
} LabelRec;

/* Types for label class methods */
typedef void (*LabelSetTextProc)();
   /* Widget widget */
   /* String text */
```

¦ Copyright IBM Corp. 1985, 1991

```
typedef String (*LabelGetTextProc)();
   /* Widget widget */

/* New fields for the Label widget class record */
typedef struct {
   LabelSetTextProc set_text;
   LabelGetTextProc get_text;
   caddr_t extension;
} LabelClassPart;

/* Full class record declaration */
typedef struct_LabelClassRec {
   CoreClassPart core_class;
   LabelClassPart label_class;
} LabelClassRec;

/* Class record variable */
extern LabelClassRec labelClassRec;

#define LabelInheritSetText((LabelSetTextProc)_XtInherit)
#define LabelInheritGetText((LabelGetTextProc)_XtInherit)
#endif LABELP_H
```

To accommodate operating systems with file name length restrictions, the name of the private **.h** file is the first nine characters of the widget class followed by a capital **P**. For example, the private **.h** file for the Constraint widget is **ConstrainP.h**.

¦ Copyright IBM Corp. 1985, 1991

4.5.3 Using Widget Subclassing in .c Files

The **.c** file for a widget contains the structure initializer for the class record variable. This initializer contains the following parts:

Class informatio

Examples: *superclass, class_name, widget_size, class_initialize, class_inited*

Data constant

Examples: *resources* and *num_resources, actions* and *num_actions, visible_interest, compress_motion, compress_exposure, version*

Widget operation

Examples: *initialize, realize, destroy, resize, expose, set_values, accept_focus*, and any operations specific to the widget.

The superclass field points to the superclass **WidgetClass** record. For direct subclasses of the generic Core widget, superclass should be initialized to the address of the **widgetClassRec** structure. The superclass is used for class chaining operations and for inheriting or enveloping the operations of a superclass. See "Chaining Superclass Operations" in topic 4.5.4, "Inheriting Superclass Operations" in topic 4.5.6, and "Calling Superclass Operations" in topic 4.5.7.

Some of the fields are:

The *class_name* which contains the text name for this class used by the resource manager. For example, the Label widget has the string **Label**. More than one widget class can use the same text class name.

The *widget_size* which is the size of the corresponding widget structure, not the size of the class structure.

The *version* which indicates the toolkit version number. This field is used at run time to check consistency of the Toolkit and widgets in an application. It should be set to the value returned by **XtVersion** in the widget class initialization.

To run widgets that are backwards compatible with previous Intrinsic versions, use the value **XtVersionDontCheck**. This value turns off version checking for those widgets.

The *extension* which is used for upwards compatibility. If you add additional fields to class parts, all subclass structure layout change and complete recompilation is required. To avoid recompilation, an extension field at the end of each class part can point to a record that contains any additional class information that is required.

The following is a compressed version of the **.c** file for the Label widget. (The resources table is in "Resource Management" in topic 4.14.)

```
/* Resources specific to Label */

#define XtRJustify "Justify"
static XtResource resources[]  = {
   {XtNforeground, XtCForeground, XtRPixel, sizeof(Pixel),
```

¦ Copyright IBM Corp. 1985, 1991

```
    XtOffset(LabelWidget, label.foreground), XtRString, XtDefaultForeground,
  {XtNfont,  XtCFont, XtRFontStruct, sizeof(XFontStruct*),
    XtOffset(LabelWidget, label.font),XtRString, XtDefaultFont},
  {XtNlabel, XtCLabel, XtRString, sizeof(String),
    XtOffset(LabelWidget, label.label), XtRString, NULL},
                          .
                          .
                          .
}
/* Forward declarations of procedures */

static void ClassInitialize();
static void Initialize();
static void Realize();
static void SetText();
static void GetText();
                          .
                          .
                          .

/* Class record constant */
LabelClassRec labelClassRec = {
  {


/* Core class fields */
/* superclass               */   (WidgetClass) &widgetClassRec,
/* class_name               */   "Label",
/* widget_size              */   sizeof(LabelRec),
/* class_initialize         */   ClassInitialize,
/* class_part_initialize    */   NULL,
/* class_inited             */   False,
/* initialize               */   Initialize,
/* initialize_hook          */   NULL,
/* realize                  */   Realize,
/* actions                  */   NULL,
/* num_actions              */   0,
/* resources                */   resources,
/* num_resources            */   XtNumber(resources),
/* xrm_class                */   NULLQUARK,
/* compress_motion          */   True,
/* compress_exposure        */   True,
/* compress_enterleave      */   True,
/* visible_interest         */   False,
/* destroy                  */   NULL,
/* resize                   */   Resize,
/* expose                   */   Redisplay,
/* set_values               */   SetValues,
/* set_values_hook          */   NULL,
/* set_values_almost        */   XtInheritSetValuesAlmost,
/* get_values_hook          */   NULL,
/* accept_focus             */   NULL,
/* version                  */   XtVersion,
/* callback_offsets         */   NULL,
/* tm_table                 */   NULL,
/* query_geometry           */   XtInheritQueryGeometry,
/* display_accelerator      */   NULL,
/* extension                */   NULL,

  },
```

¦ Copyright IBM Corp. 1985, 1991

```
  {


/* Label class fields */
/* get_text                  */   GetText,
/* set_text                  */   SetText,
/* extension                 */   NULL,

  }
};

/* Class record pointer */
WidgetClass labelWidgetClass = (WidgetClass) &labelClassRec;

/* New method access routines */
void Label SetText (widget, text)
  Widget widget;
  String text;
{
  Label WidgetClass lwc = (Label WidgetClass) XtClass(widget);
  XtCheckSubclass(widget, labelWidgetClass, NULL);
  *(lwc->label_class.set_text)(widget, text)
}

/* Private procedures */
                              .
                              .
                              .
```

Subtopics
4.5.3.1 Related Functions

¦ Copyright IBM Corp. 1985, 1991

4.5.3.1 Related Functions

XtClas

XtSuperClas

XtIsSubclas

XtCheckSubclas

¦ Copyright IBM Corp. 1985, 1991

4.5.4 Chaining Superclass Operations

Some fields defined in the widget class structure are self-contained and independent of the values for these fields defined in superclasses. Among these are:

class_name

accept_focus

class_initialize

compress_motion

widget_size

compress_exposure

realize

compress_enterleave

visible_interest

set_values_almost

resize

tm_table

expose

version

Some fields defined in the widget class structure are accessed only after their corresponding superclass value has been accessed. This is called downward superclass chaining. In this case, the invocation of a single operation first accesses the Core
class, then the subclass, and so on down the class chain to the widget class of the widget. These superclass-to-subclass fields are:

class_part_initialize

get_values_hook

initialize_hook

initialize

set_values_hook

set_values

resources

¦ Copyright IBM Corp. 1985, 1991

In addition, for subclasses of **Constraint**, the resources field of the **ConstraintClassPart** structure is chained from the **Constraint** class down to the subclass.

Some fields defined in the widget class structure are accessed before their corresponding superclass value has been accessed. This is called upward superclass chaining. In this case, the invocation of a single operation actually first accesses the widget class, then its superclass, and so on up the class chain to the Core class. The subclass-to-superclass fields are:

destroy
actions.

¦ Copyright IBM Corp. 1985, 1991

4.5.5 Initializing a Widget Class

Many class records can be initialized completely at compile time. In some cases, however, a class may need to register type converters or perform other kinds of "one-time" initializations.

The C language does not have initialization procedures that are invoked automatically when a program starts up. Thus, a widget class can declare a *class_initialize* procedure that will be called (once) automatically by the Toolkit. A class initialization procedure pointer is of type **XtProc**.

```
typedef void (*XtProc) ();
```

Specify **NULL** in the *class_initialize* field to indicate that a widget class does not have a class initialization procedure.

In addition to class initializations, some widget classes must must perform additional initializations for fields in their part of the class record. These initializations are performed in the *class_part_init* procedure, which is stored in the *class_part_initialize* field. The *class_part_initialize* procedure pointer is of type **XtWidgetClassProc**:

```
typedef void (*XtWidgetClassProc)(WidgetClass);
```

During class intialization, the class part initialization procedure for the class and its superclasses are called in superclass-to-subclass order on the class record.

These procedures perform necessary dynamic initializations to the part of the record for their class. The most common procedure is the resolution of any inherited methods defined in the class. For example, if a widget class *C* has superclasses Core, Composite, *A*, and *B*, the class record for *C* is passed first to the *class_part_initialize* record of Core. This resolves any inherited Core methods and compiles the textual representations of the resource list and action table that are defined in the class record. Next, the Composite *class_part_initialize* procedure initializes the composite part of *C*'s class record. Finally, the *class_part_initialize* procedures for *A, B,* and *C* are called in order. See "Inheriting Superclass Operations" in topic 4.5.6.

Specify **NULL** in the *class_part_initialize* field for classes that do not define new class fields or that do not need extra processing for their class fields.

All widget classes must begin with *class_inited* set to **False**, whether or not they have a class initialization procedure.

The first time a widget of a class is created, **XtCreateWidget** ensures that the widget class and all superclasses are initialized in superclass to subclass order by checking that each *class_inited* field is **False**.

Then, the Instrinsics set the *class_inited* field to **True**. After the one-time initialization, a class structure is constant.

The following is the class initialization procedure for Label.

```
static void ClassInitialize()
{
```

¦ Copyright IBM Corp. 1985, 1991

```
    XtQEleft   = XrmStringToQuark("left");
    XtQEcenter = XrmStringToQuark("center");
    XtQEright  = XrmStringToQuark("right");

    XtAddConverter(XtRString, XtRJustify, CvtStringToJustify, NULL, 0);

}
```

A class is initialized the first time a widget of that class, or any subclass, is created. If the class initialization procedure registers type converters, these type converters are not available until the first widget is created. See "Converting Resources" in topic 4.14.3.

¦ Copyright IBM Corp. 1985, 1991

4.5.6 Inheriting Superclass Operations

A widget class can use any of the self-contained operations of its superclass rather than implementing its own code. The inherited operations of the superclass most frequently used include:

```
expos
realiz
insert_chil
delete_chil
geometry_manage
set_values_almost
```

To inherit an operation *xyz*, specify the constant **XtInherit***Xyz* in your class record. Every class that declares a new procedure in its widget class part must provide for inheriting the procedure in its class_part_initialize procedure.

The special chained operations *initialize*, *set_values* and *destroy*, which are declared in the Core record, do not have inherit procedures. Widget classes that do nothing beyond what their superclass does, specify **NULL** for chained procedures in their class records.

Inheriting compares the value of the field with a known, special value. If a match occurs, it copies the superclass value for that field. This special value is usually the Intrinsics internal value **_XtInherit** cast to the appropriate type. (**_XtInherit** issues an error message if it is called directly.)

For example, the Composite class private include file contains these definitions:

```
#define XtInheritGeometryManager ((XtGeometryHandler) _XtInherit)
#define XtInheritChangeManaged ((XtWidgetProc) _XtInherit)
#define XtInheritInsertChild ((XtArgsProc) _XtInherit)
#define XtInheritDeleteChild ((XtWidgetProc) _XtInherit)
```

The Composite *class_part_initialize* procedure begins as follows:

```
static void CompositeClassPartInitialize(widgetClass)
     WidgetClass widgetClass;
{
     register CompositeWidgetClass wc = (CompositeWidgetClass)widgetClass;
     CompositeWidgetClass super = (CompositeWidgetClass)wc->core_class.superc

     if (wc->composite_class.geometry_manager == XtInheritGeometryManager) {
         wc->composite_class.geometry_manager = super->composite_class.geometr
         manager;
     }

     if (wc->composite_class.change_managed == XtInheritChangeManaged) {
         wc->composite_class.change_managed = super->composite_class.change_
         managed;
     }
                              .
                              .
                              .
```

The defined inherit constants for Core are:

¦ Copyright IBM Corp. 1985, 1991

XtInheritRealize
XtInheritResize
XtInheritExpose
XtInheritSetValuesAlmost
XtInheritAcceptFocus
XtInheritDisplayAccelerator.

The defined inherit constants for Composite are:

XtInheritGeometryManager
XtInheritChangeManaged
XtInheritInsertChild
XtInheritDeleteChild.

¦ Copyright IBM Corp. 1985, 1991

4.5.7 Calling Superclass Operations

A widget class sometimes needs to call a superclass operation that usually is not chained. For example, the expose procedure for a widget might call the expose procedure of its superclass and then perform more work of its own. Composite classes with fixed children can implement *insert_child* by calling their superclass *insert_child* procedure; then, calling **XtManageChild** to add the child to the managed list.

The class procedure should call its own superclass procedure, not the superclass of the widget. The class procedure should use its own class pointers and not the class pointers of the widget. This technique is referred to ***enveloping*** the operation of the superclass.

¦ Copyright IBM Corp. 1985, 1991

4.6 Instantiating Widgets

Widgets are either ***primitive*** or ***composite***. Either kind can have children widgets.

> *Primitive* widgets containing children typically instantiate their children and do not expect external clients to do so. Primitive widgets do not have general geometry management methods. Primitive widgets that instantiate children are responsible for all operations requiring downward traversal below themselves.
> *Composite* widgets are containers for an arbitrary, implementation-defined collection of children. These children may be instantiated by the composite widget itself, by other clients, or by a combination. Composite widgets contain methods for managing the geometry (layout) of any child widget.

The Intrinsics provide management mechanisms for constructing and interfacing between composite widgets, their children, and other clients. The Intrinsics also recursively perform many operations, such as realization and destruction, on composite widgets and all of their children.

While a composite widget can, in unusual circumstances, have no children, they usually have at least one.

Pop-up children can be attached to any widget, regardless of class. Each pop-up child has a window that is a child of the root window so that the pop-up window is not clipped.

Widgets with no children of any kind are ***leaves*** of a widget tree. Widgets with one or more children are ***intermediate nodes*** of a tree. The shell widget returned by **XtAppCreateShell** is the ***root*** of a widget tree.

The normal children of the widget tree (including pop-ups) define the associated X-window tree.

A widget tree is manipulated by several Intrinsics functions:

> **XtRealizeWidget**, which traverses the tree downward and recursively realizes all pop-up widgets and children of composite widgets.
> **XtDestroyWidget**, which traverses the tree downward and destroys all pop-up widgets and children of composite widgets.
> **XtMakeGeometryRequest**, which traverses the tree one level upward and calls the geometry manager responsible for the child geometry of a widget.
>
> The functions that get and modify resources traverse the tree upward to determine the inheritance of resources from the ancestors of a widget.

To facilitate up-traversal of the widget tree, each widget has a pointer to its parent widget. However, **XtAppCreateShell** returns Shell widgets with a parent pointer of **NULL**.

To facilitate down-traversal of the widget tree, each composite widget has a pointer to an array of children widgets. This array includes all normal children created, not just the subset of children that are managed by the composite geometry manager of the widget. In addition, every widget has a pointer to an array of pop-up children widgets.

¦ Copyright IBM Corp. 1985, 1991

Subtopics
4.6.1 Initializing the Toolkit
4.6.2 Loading the Resource Database

¦ Copyright IBM Corp. 1985, 1991

4.6.1 Initializing the Toolkit

Before an application can call any of the Intrinsics functions, it must initialize the Toolkit by using:

XtToolkitInitialize, which initializes the Toolkit internals.

XtCreateApplicationContext, which initializes the per application state.

XtDisplayInitialize or **XtOpenDisplay**, which initialize the per display state.

XtAppCreateShell, which creates the initial widget.

Multiple instances of Toolkit applications can be implemented by a single program in a single address space. Each instance must be able to read input and dispatch events independently of any other instance.

Applications may need multiple display connections or the same widgets on multiple screens. To achieve this, the Intrinsics define application contexts, which provide the information necessary to distinguish one application instance from another. A list of **Display** pointers for that application is the major component of an application context. The application context type **XtAppContext** is opaque to clients.

Subtopics
4.6.1.1 Related Functions

¦ Copyright IBM Corp. 1985, 1991

4.6.1.1 Related Functions

XtToolkitInitializ

XtCreateApplicationContex

XtDestroyApplicationContex

XtWidgetToApplicationContex

XtDisplayInitializ

XtOpenDispl

XtCloseDispla

¦ Copyright IBM Corp. 1985, 1991

4.6.2 Loading the Resource Database

The **XtDisplayInitialize** function loads the application's resource database for this display, host, and application combination. Each resource database is kept on a per-display basis, and is loaded from five sources, if they exist, in the following order:

1. Application-specific class resource file on the local host
2. Application-specific user resource file on the local host
3. Resource property on the server or user preference resource file on the local host
4. Per-host user environment resource file on the local host
5. Application command line (**argv**).

The application-specific resource file name is constructed from the class name of the application and points to a site-specific resource file that is installed by the site manager, usually when the application is installed. The application resource file is **/usr/lpp/X11/appdefaults/***class*, where *class* is the application class name. This file should be provided by the application developer and may be required for the application to run properly.

The application-specific user resource file name is constructed from the class name of the application and points to a user-specific resource file. This file is owned by the application and typically stores user customizations. This file name is constructed by using the value of the user's **XAPPLRESDIR** environment variable and appending *class* to it, where *class* is the application name. If this environment variable is not defined, the value defaults to the user's home directory. If the resource file exists, it is merged into the resource database. The file may be provided with the application or constructed by the user.

The server resource file is the contents of the X Server RESOURCE_MANAGER property as returned by **XOpenDisplay**. The server resource file is constructed entirely by the user and contains both display-independent and display-specific user preferences. If the RESOURCE_MANAGER property does not exist, the resource file in the user's home directory is used. This file is usually called **.Xdefaults**. If the resource file exists, it is merged into the resource database.

The user's environment resource file name is constructed by using the value of the user's *XENVIRONMENT* variable for the full path of the file. If the user's environment resource file exists, it is merged into the resource database. (This filename is user and host-specific.)

If the *XENVIRONMENT* variable does not exist, **XtDisplayInitialize** searches the user's home directory for the **.Xdefaults-***host* file, where *host* is the name of the system where the application is running. If this file exists, it is merged into the resource database.

The environment resource file should contain process-specific resource specifications intended to supplement the user-preference specifications in the server resource file.

Use **XtDatabase** to obtain the resource database for a particular display. This routine returns the fully merged resource database built by **XtDisplayInitialize**. See "XtDatabase" in topic 4.17.71.

¦ Copyright IBM Corp. 1985, 1991

4.7 Creating Widgets

The creation of widget instances is a three-phase process:

1. The widgets are allocated and initialized with resources and are optionally added to the managed subset of their parent.

2. All composite widgets are notified of their managed children in a bottom-up traversal of the widget tree.

3. The widgets create windows that then get mapped.

To start the first phase, the application calls **XtCreateWidget** for all its widgets and adds them, as appropriate, to the managed set of their parent by calling **XtManageChild**. To avoid an O(n(2)) creation process where each composite widget lays itself out each time a widget is created and managed, parent widgets are not notified of changes in their managed set during this phase.

After all widgets have been created, the application calls **XtRealizeWidget** on the top-level widget to start the second and third phases. **XtRealizeWidget** first recursively traverses the widget tree in a post-order (bottom-up) traversal and then notifies each composite widget that has one or more managed children through its *change_managed* procedure.

Notifying a parent about its managed set involves geometry layout and, possibly, geometry negotiation. A parent must deal with constraints on its size that are imposed from above, as when a user specifies the application window size. A parent also deals with suggestions made from below, as when a primitive child computes its preferred size. Any clash between the two can cause geometry changes to ripple in both directions through the widget tree. The parent may force some of its children to change size and position and may issue geometry requests to its own parent to accommodate all its children better. Until this process is settled, placement on the screen is uncertain.

Consequently, to avoid unnecessary requests to the X Server to move windows after creation, no windows are actually created in the first and second phases.

Finally, **XtRealizeWidget** starts the third phase by making a pre-order (top-down) traversal of the widget tree, allocates an X-Windows window to each widget by means of its realize procedure, and maps the managed widgets.

Subtopics

¦ Copyright IBM Corp. 1985, 1991

4.7.1 Creating and Merging Argument Lists

Many of the Intrinsics routines need to be passed pairs of resource names and values, called an ***argument list***. These are passed as an **ArgList** structure, which contains:

```
typedef type XtArgVal;

typedef struct {
       String name;
       XtArgVal value;
} Arg, *ArgList;
```

In the preceding definition, *type* is a C language type large enough to contain a pointer to a function or a parameter of type **caddr_t**, **char ***, **long**, or **int ***.

If the size of the resource is less than, or equal to, the size of an **XtArgVal**, the resource value is stored directly in *value*. Otherwise, *value* is a pointer to the resource value.

For more information on setting values in an **ArgList**, see "XtSetArg" in topic 4.17.160 For more information on merging two **ArgList** structures, see "XtMergeArgLists" in topic 4.17.119.

¦ Copyright IBM Corp. 1985, 1991

4.7.2 Creating a Widget Instance

The **XtCreateWidget** routine creates an instance of a widget. This routine performs many of the boilerplate operations of widget creation. For more information about this routine, see "XtCreateWidget" in topic 4.17.69.

¦ Copyright IBM Corp. 1985, 1991

4.7.3 Creating an Application Shell Instance

Applications can have multiple top-level widgets, which can be on different screens. To create several independent windows, applications use the **XtAppCreateShell**, which creates a top-level widget that is the root of a widget tree.

To create multiple top-level shells within a single logical application, use one of the following methods:

Designate one shell as the real top-level shell and create the other as pop-up children of the first with **XtCreatePopupShell**.

Use this method when you need to designate a main window. It leads to resource specifications as the following:

xmail.geometry:... (main window)

xmail.read.geometry:... (read window)

xmail.compose.geometry:... (compose window)

Have all shells as pop-up children of an unrealized top-level shell

Use this method when you do not need to create a main window. It leads to resource specifications as the following:

xmail.headers.geometry:... (headers window)

xmail.read.geometry:... (read window)

xmail.compose.geometry:... (compose window)

For additional information on creating multiple top-level shells, see "XtAppCreateShell" in topic 4.17.23.

¦ Copyright IBM Corp. 1985, 1991

4.7.4 Initializing a Widget Instance

An initialization procedure performs the following:

Allocates space and copy any resources that are referenced by address For example, if a widget has a field that is a string (**char** *), it cannot depend upon the characters at that address to remain constant, but must dynamically allocate space for the string and copy it to the new space.

Note: Do not allocate space for or copy callback lists.

Computes values for unspecified resource fields. For example, i width and height are zero, the widget computes a width and height based on other resources. This is the only time a widget can directly assign its own width and height.

Computes values for uninitialized non-resource fields that are derive from resource fields. For example, **GCs** that the widget uses are derived from resources like background, foreground, and font.

An initialization procedure can also check certain fields for internal consistency, such as a specification of a colormap for a depth that does not support that colormap.

Initialization procedures are called in superclass-to-subclass order. Most of the initialization code for a specific widget class deals with fields defined in that class and not with fields defined in its superclasses.

Specify **NULL** for the initialize field in the class record if a subclass does not need an initialization procedure.

Sometimes a subclass may want to overwrite values filled in by its superclass. In particular, size calculations of a superclass are often incorrect for a subclass. In this case, the subclass must modify or recalculate fields declared and computed by its superclass. For example, a subclass can visually surround its superclass display. In this case, the width and height calculated by the superclass initialize procedure are too small and need to be incremented by the size of the surround. The subclass needs to know if its superclass size was calculated by the superclass or was specified explicitly. All widgets must place themselves into whatever size is explicitly given, but widgets should compute a reasonable size if no size is requested.

A subclass compares the difference between a specified size and a size computed by a superclass by checking the *request* and *new* parameters in **XtInitProc**.

The initialize procedure pointer in a widget class is of type **XInitProc**:

```
typedef void (*XtInitProc)(Widget, Widget);
     Widget request;
     Widget new;
```

The *request* specifies the widget with resource values as requested by the argument list, the resource database, and the widget defaults. The *new* specifies a widget with the new values, both resource and nonresource, that are allowed. A subclass initialize procedure compares the values to

¦ Copyright IBM Corp. 1985, 1991

resolve potential conflicts. See "XtInitProc" in topic 4.17.101.

In this example, the subclass with the visual surround can see if the width and height in the request widget are zero. If so, the subclass adds its surround size to the width and height fields in the new widget. If not, the subclass must use the size originally specified.

The new widget becomes the actual widget instance record. Therefore, if the initialization procedure needs to call any routines that operate on a widget, it should specify "new" as the widget instance. The request widget should never be modified.

Subtopics
4.7.4.1 Initializing a Constraint Widget Instance
4.7.4.2 Initializing Non-widget Data

¦ Copyright IBM Corp. 1985, 1991

4.7.4.1 Initializing a Constraint Widget Instance

The constraint initialize procedure is of type **XtInitProc**. (See "XtInitProc" in topic 4.17.101.) The values passed to the parent constraint initialization procedure are the same values passed to the class widget initialization procedure of the child widget.

The constraint initialize procedure should compute any constraint fields derived from constraint resources. It can make further changes to the widget to make the widget conform to the specified constraints, for example, changing the size or position of the widget.

Specify **NULL** for the *initialize* field of the **ConstraintClassPart** in the class record if a constraint class does not need a constraint initialization procedure.

¦ Copyright IBM Corp. 1985, 1991

4.7.4.2 Initializing Non-widget Data

The *initialize_hook* procedure allows a widget instance to initialize non-widget data using information from the specified argument list. For example, the Text widget has subparts that are not widgets, but these subparts have resources that can be specified by the resource file or an argument list. The *initialize_hook* procedure is of type **XtArgsProc**:

```
typedef void (*XtArgsProc)(Widget, Arglist, Cardinal*);
     Widget  widget;
     ArgList  args;
     Cardinal  *num_args;
```

See "XtArgsProc" in topic 4.17.42. See also "XtGetSubresources" in topic 4.17.96.

¦ Copyright IBM Corp. 1985, 1991

4.7.5 Realizing Widgets

All the Instrinsics routines and all widget routines should work with realized or unrealized widgets. To realize a widget instances, use the **XtRealizeWidget**. (See "XtRealizeWidget" in topic 4.17.142.) The widget procedure must create a window for the widget. In addition, some widget procedures, such as **set_values**, may choose to operate differently after the widget has been realized.

XtCreateWidget, **XtRealizeWidget**, **XtManageChildren**, **XtUnmanageChildren**, and **XtDestroyWidget** maintain the following invariants:

If a widget is realized, then all managed children of the widget ar realized.
If a widget is realized, then all managed children of the widget tha are *mapped_when_managed* are mapped.

To determine whether a widget has been realized, use the **XtIsRealized** routine. For more information on this routine, see "XtIsRealized" in topic 4.17.107.

Subtopics
4.7.5.1 Creating a Window for a Widget Instance

¦ Copyright IBM Corp. 1985, 1991

4.7.5.1 Creating a Window for a Widget Instance

A widget class can inherit its realize procedure from its superclass during class initialization. The realize procedure defined for Core calls **XtCreateWindow** which passes the *value_mask* and with **windowClass** and *visual* set to **CopyFromParent**. Both **CompositeWidgetClass** and **ConstraintWidgetClass** inherit this realize procedure, and most new widget subclasses can do the same. See "Inheriting Superclass Operations" in topic 4.5.6.

The most common noninherited realize procedures set *bit_gravity* in the mask, set attributes to the appropriate value, and then create the window. For example, Label sets *bit_gravity* to **WestGravity, CenterGravity** or **EastGravity**. Consequently, shrinking a Label moves the bits appropriately, and no **Expose** event is needed for repainting.

If a composite widget wants to realize its children in a particular order, typically to control the stacking order, it calls **XtRealizeWidget** on its children in the appropriate order from within its own realize procedure.

Rather than call **Xlib XCreateWindow** directly, a realize procedure should call the Toolkit analog **XtCreateWindow**. This routine, which simplifies the creation of windows for widgets, evaluates the following fields in the Core widget structure:

x
y
width
height
depth
screen
parent -> core.window

Widgets that have children but are not a subclass of the *compositeWidgetClass* are responsible for calling the **XtRealizeWidget** routine for their children, usually from within the realize procedure.

Note: Because realize is not a chained operation, the widget class realize procedure must update the **XSetWindowAttributes** structure with all the appropriate fields from non-Core superclasses.

¦ Copyright IBM Corp. 1985, 1991

4.7.6 Obtaining Window Information

The Core widget definition contains the screen and window IDs. The window field may be **NULL** for a while. See "Creating Widgets" in topic 4.7 and "Realizing Widgets" in topic 4.7.5.

The display pointer, the parent widget, the screen pointer, and the window, of a widget are returned by the following macros, which take a widget and return the specified value:

Display *XtDisplay(*widget*)**
    Widget *widget;*

Widget XtParent(*widget*)
    Widget *widget;*

Screen *XtScreen(*widget*)**
    Widget *widget;*

Window XtWindow(*widget*)
     Widget *widget;*

Several window attributes are cached locally in the widget. Thus, the widgets can be set by the Resource Manager and **XtSetValues** as well as **XtCreateWindow** or routines that derive structures from these values. Examples of such structures are *depth* for deriving pixmaps and *background_pixel* for deriving **GCs**.

The *x, y, width, height,* and *border_width* window attributes are available to geometry managers. These fields are maintained synchronously inside the Toolkit. When an **XConfigureWindow** is issued on the widget window at the request of the parent, these values are updated immediately rather than whenever the server generates a **ConfigureNotify** event. In fact, most widgets do not have **SubstructureNotify** turned on. This ensures that all geometry calculations are based on the internally consistent Toolkit, rather than on either:

    Inconsistencies updated by asynchronous **ConfigureNotify** events

    Wasteful consistencies, which slow the process because geometr managers ask the server for window sizes each time they need to layout their managed children. See "Managing Widget Geometry" in topic 4.13 for further information.

Subtopics
4.7.6.1 Unrealizing Widgets

¦ Copyright IBM Corp. 1985, 1991

4.7.6.1 Unrealizing Widgets

After widgets have been realized, they can also be unrealized. To destroy the windows associated with a widget and its descendants, use the **XtUnrealizeWidget** routine. For more information on destroying the windows associated with widgets, see "XtUnrealizeWidget" in topic 4.17.184.

¦ Copyright IBM Corp. 1985, 1991

4.7.7 Destroying Widgets

To destroy widgets, the Toolkit can:

Destroy all the pop-up children of the widget being destroyed an destroy all children of composite widgets

Remove and unmap the widget from its paren

Call the callback procedures that have been registered to trigger whe the the widget is destroyed

Minimize the number of things a widget has to deallocate whe destroyed

Minimize the number of **XDestroyWindow** calls.

To destroy a widget instance, use the **XtDestroyWidget** routine. For more information on destroying widget instances, see "XtDestroyWidget" in topic 4.17.74.

Subtopics
4.7.7.1 Adding and Deleting Destroy Callbacks
4.7.7.2 Deallocating Dynamic Constraint Data
4.7.7.3 Exiting an Application

¦ Copyright IBM Corp. 1985, 1991

4.7.7.1 Adding and Deleting Destroy Callbacks

When an application needs to perform additional processing during the destruction of a wdiget, it should register a destroy callback procedure for the widget. The destroy callback uses the mechanism described in "Using Callbacks" in topic 4.7.8. The destroy callback list is identified by the resource name **XtNdestroyCallback**.

The following example calls **XtAddCallback** to add to a widget the application-supplied destroy callback procedure, **ClientDestroy**, which has data *client_data*.

XtAddCallback (*widget***, XtNdestroyCallback, +**
ClientDestroy, *client_data***)**

Similarly, the next example removes the application-supplied destroy callback procedure **ClientDestroy** by calling **XtRemoveCallback**.

XtRemoveCallback (*widget***, XtNdestroyCallback, +**
ClientDestroy, *client_data***)**

The **ClientDestroy** procedure in both of the preceding examples is of type **XtCallbackProc**. For more information on this data type, see "XtCallbackProc" in topic 4.17.52.

¦ Copyright IBM Corp. 1985, 1991

4.7.7.2 Deallocating Dynamic Constraint Data

The constraint destroy procedure type is **XtWidgetProc**.

```
typdef void (*XtWidgetProc)(Widget);
```

The destroy procedure is called for a widget whose parent is a subclass of *constraintWidgetClass*. The constraint destroy procedures are called in subclass-to-superclass order, starting at the parent of the widget and ending at *constraintWidgetClass*. Therefore, a constraint destroy procedure of a parent should deallocate only storage that is specific to the constraint subclass and not the storage allocated by any of its superclasses. Use the NULL entry for the constraint destroy procedure for the parent not requiring deallocation of constraint storage.

¦ Copyright IBM Corp. 1985, 1991

4.7.7.3 Exiting an Application

Use **XtDestroyApplicationContext** to terminate all toolkit applications. Then, exit normally. Or, use **XtUnmapWidget** on each top-level shell widget.

¦ Copyright IBM Corp. 1985, 1991

4.7.8 Using Callbacks

Applications and other widgets (clients) often want to register a procedure with a widget that is called under certain conditions. For example, when a widget is destroyed, every procedure in its *destroy_callbacks* list is called to notify clients of its impending destruction.

Every widget has a *destroy_callbacks* list. Additional callback lists can be defined as needed. For example, the Command widget has a callback list to notify clients when the button has been activated. Callback procedure fields for use in callback lists are of type **XtCallbackProc**. For a definition of this data type, see "XtCallbackProc" in topic 4.17.52.

To pass a callback list as an argument in a call to **XtCreateWidget**, **XtSetValues**, or **XtGetValues**, a client specifies the address of a null-terminated array of type **XtCallbackList**, which is described under "XtCallbackList" in topic 4.17.48.

For example, the callback list for procedures **A** and **B** with client data **clientDataA** and **clientDataB**, respectively looks like:

```
static XtCallbackRec callbacks[] = {
    {A, (caddr_t) clientDataA},
    {B, (caddr_t) clientDataB},
    {(XtCallbackProc) NULL, (caddr_t) NULL}
};
```

Although callback lists are passed by address in argument lists, the Intrinsics are aware of callback lists. Application-specific *initialize* and *set_values* procedures should not allocate memory for the callback list, as the Intrinsics do this automatically by using a different structure for their internal representation.

Whenever a widget contains a callback list for use by clients, it also exports in its public **.h** file the resource name of the callback list. Applications and client widgets never access callback list fields directly. Instead, they identify the desired callback list using the exported resource name. All callback manipulation routines check that the requested callback list is implemented by the widget.

For the Intrinsics to find and correctly handle callback lists, the lists should always be declared with a resource type of **XtRCallback**.

To perform operations involving callback lists, use the following routines:

XtAddCallback	Adds a callback procedure to a widget's callback list. See "XtAddCallback" in topic 4.17.7.
XtAddCallbacks	Adds a list of callback procedures to a widget's callback list. See "XtAddCallbacks" in topic 4.17.8.
XtRemoveCallback	Deletes a callback procedure from a widget's callback list. See "XtRemoveCallback" in topic 4.17.147.
XtRemoveCallbacks	Deletes a list of callback procedures from a widget's callback list. See "XtRemoveCallbacks"

¦ Copyright IBM Corp. 1985, 1991

in topic 4.17.148.

XtRemoveAllCallbacks	Deletes all callback procedures from a widget's callback list See "XtRemoveAllCallbacks" in topic 4.17.146. and frees all storage associated with the callback list.
XtCallCallbacks	Calls the procedures in a widget's callback list. See "XtCallCallbacks" in topic 4.17.53.
XtHasCallbacks	Determines the status of a widget's callback list. See "XtHasCallbacks" in topic 4.17.99.

¦ Copyright IBM Corp. 1985, 1991

4.8 Using Composite Widgets

Composite widgets are a subclass of **compositeWidgetClass**. Composite widget functions are implemented directly by the widget class or indirectly by Intrinsics functions. Composite widgets can have an arbitrary number of children and consequently control more than primitive widgets. In general, composite widgets handle:

Overall management of children from creation to destructio

Destruction of descendants when the composite widget is destroye

Physical arrangement, or geometry management, of a displayable subse of managed children

Mapping and unmapping of a subset of the managed children

Overall management is handled by the generic procedures **XtCreateWidget** and **XtDestroyWidget**. **XtCreateWidget** adds children to a parent by calling the *insert_child* procedure of the parent. **XtDestroyWidget** removes children from a parent by calling the *delete_child* procedure of the parent and also ensures that all children of a destroyed composite widget get destroyed.

Only a subset of the total number of children is actually managed by the geometry manager and possibly visible. For example, a multibuffer composite editor widget can allocate one child widget for each file buffer while only displaying a small number of the existing buffers. Windows in this displayable subset are ***unmanaged windows*** and enter into geometry manager calculations. The other children are ***unmanaged windows*** and are not mapped.

Children are added to the managed set with **XtManageChild**, **XtManageChildren** and removed from the managed set by using **XtUnmanageChild**, and **XtUnmanageChildren**. These procedures notify the parent to recalculate the physical layout of its children by calling the *change_managed* procedure of the parent. The **XtCreateManagedWidget** function calls **XtCreateWidget** and **XtManageChild** on the result.

Most managed children are mapped, but some widgets can be in a state where they take up physical space, but do not show anything. Managed widgets are not mapped automatically if the *map_when_managed* field is **False**. The default for *map_when_managed* is **True**. This field is changed by using **XtSetMappedWhenManaged**.

Each composite widget class has a geometry manager responsible for calculating where the managed children appear within the window of the composite widget. Geometry management techniques fall into four classes:

Fixed boxes have a fixed number of children created by the parent. All of these children are managed and none make geometry manager requests.

Homogeneous boxes treat all children equally and apply the same geometry constraints to each child. Many clients insert and delete widgets freely.

Heterogeneous boxes place each child in a specific location. This location is not usually specified in pixels because the window can be resized, but is expressed in terms of the relationship between a child and the parent or between the child and other specific children.

¦ Copyright IBM Corp. 1985, 1991

Heterogeneous boxes are usually subclasses of **Constraint**.

Shell boxes have only one child. This child is exactly the size of the shell. The geometry manager must communicate with the window manager if it exists. The box must also accept **ConfigureNotify** events when the window size is changed by the window manager.

Subtopics

¦ Copyright IBM Corp. 1985, 1991

4.8.1 Verifying the Class of a Composite Widget

To determine if a given widget is a subclass of **Composite**, use the **XtIsComposite** function. This function is equivalent to **XtIsSubclass** with **compositeWidgetClass** specified. (See "XtIsComposite" in topic 4.17.105.)

¦ Copyright IBM Corp. 1985, 1991

4.8.2 Adding Children to a Composite Widget

To add a child to the list of children of a parent, the **XtCreateWidget** function calls the *insert_child* class routine of the parent. The *insert_child* procedure pointer in a composite widget is of type **XtWidgetProc**:

```
typedef void(*XtWidgetProc)(Widget);
```

Most composite widgets inherit the operation of their superclass. The *insert_child* routine of the composite widget calls the *insert_position* procedure and inserts the child at the specified location.

Some composite widgets define their own *insert_child* routine so that they can order their children in some convenient way, create companion controller widgets for a new widget, or limit the number or type of their children widgets.

If there is not enough room to insert a new child in the children array (if *num_children* = *num_slots*), the *insert_child* procedure must first reallocate the array and update *num_slots*. The *insert_child* procedure calculates a location for the child, inserts the child in the location, and increments the *num_children* field.

¦ Copyright IBM Corp. 1985, 1991

4.8.3 Inserting Children in a Specific Order

Instances of composite widgets must specify the order in which their children are kept. For example, an application may require a set of command buttons in some logical order grouped by function or buttons representing file names in alphabetical order.

The *insert_position* procedure pointer in a composite widget instance is of type **XtOrderProc**:

```
typedef Cardinal(*XtOrderProc)(Widget);
     Widget widget;
```

Composite widgets that allow clients to order their children are usually homogeneous boxes. These widgets call the *insert_position* procedure of the widget instance from the *insert_child* procedure of the class to determine where a new child is placed in the child array of the composite widget. A client of a composite class can apply different sorting criteria to widget instances of the class, passing in a different *insert_position* procedure when it creates each composite widget instance.

The return value of the *insert_position* procedure indicates how many children go before the widget.

When a zero is returned, the widget goes before all other children

When *num_children* is returned, the widget goes after all other children.

The default *insert_position* returns *num_children*. This default can be overridden by the resource list of a widget or by the argument list provided when the composite widget is created.

¦ Copyright IBM Corp. 1985, 1991

4.8.4 Deleting Children of Composite Widgets

To remove a child from the children array of a parent, the **XtDestroyWidget** function calls the *delete_child* procedure of the composite parent. The *delete_child* procedure pointer is of type **XtWidgetProc**:

```
typedef void(*XtWidgetProc)(Widget);
```

Most widgets inherit the *delete_child* procedure from their superclass. Composite widgets that create companion widgets define a *delete_child* procedure to remove these companion widgets.

¦ Copyright IBM Corp. 1985, 1991

4.8.5 Managing Children in a Managed Set

The Intrinsics provide a set of generic routines for adding widgets to the managed set of a composite widget and removing widgets from the managed set of a composite widget. These generic routines call the *change_managed* procedure of the widget. The *change_managed* procedure pointer is of type **XtWidgetProc**.

```
typedef void (*XtWidgetProc)(Widget);
```

Subtopics
4.8.5.1 Adding Children to a Managed Set
4.8.5.2 Removing Children from a Managed Set
4.8.5.3 Determining if a Widget is Managed
4.8.5.4 Controlling Widget Mapping

¦ Copyright IBM Corp. 1985, 1991

4.8.5.1 Adding Children to a Managed Set

To add a list of widgets to the geometry-managed displayable subset of its composite parent widget, the application must first create the widgets using **XtCreateWidget** and then call **XtManageChildren**. See "XtManageChildren" in topic 4.17.117.

Managing children is independent of the ordering, creating, and deleting of children. The layout routine of the parent considers children with a managed field equal to **True** and ignores all other children. Some composite widgets, especially fixed boxes, call **XtManageChild** from their *insert_child* procedure.

If the parent widget is realized, the *change_managed* procedure of the parent confirms that its set of managed children has changed. The parent can reposition and resize any of its children. The position of a child is changed by a call to **XtMoveWidget**. This function updates the *x* and *y* fields and calls **XMoveWindow** if the widget is realized.

To change the size or border width of any of its children, a composite widget calls **XtResizeWidget**. (See "XtResizeWidget" in topic 4.17.155.) This procedure first updates the **Core** fields and then calls the Xlib **XConfigureWindow** function if the widget is realized.

To add a single child to the list of managed children of a parent widget, first create the child widget with **XtCreateWidget** and then use **XtManageChild**. The **XtManageChild** function constructs a **WidgetList** of length one and calls **XtManageChildren**. (See "XtManageChild" in topic 4.17.116.)

To create and manage a child widget in a single procedure, use the **XtCreateManagedWidget** function. (See "XtCreateManagedWidget" in topic 4.17.67.) This function calls **XtCreateWidget** and **XtManageChild**.

¦ Copyright IBM Corp. 1985, 1991

4.8.5.2 Removing Children from a Managed Set

To remove children from the managed list of a parent widget, use **XtUnmanageChildren**. The specified widgets are not destroyed by this function and can be added to the managed list again at a later time. See "XtUnmanageChildren" in topic 4.17.182.

To remove a single child from the managed set of a parent, use **XtUnmanageChild**. (See "XtUnmanageChild" in topic 4.17.181.) The **XtUnmanageChild** function constructs a widget list of length one and calls **XtUnmanageChildren**.

These generic functions are low-level routines used by generic composite widget building routines. Composite widgets also can provide widget-specific high-level procedures for the creation and management of children.

¦ Copyright IBM Corp. 1985, 1991

4.8.5.3 Determining if a Widget is Managed

To determine the managed state of a given child widget, use **XtIsManaged**. **XtIsManaged** is implemented as a boolean macro and a function. Both forms return **True** if the specified child widget is managed and **False** if the child is not managed. See "XtIsManaged" in topic 4.17.106.

¦ Copyright IBM Corp. 1985, 1991

4.8.5.4 Controlling Widget Mapping

A managed widget is usually mapped. The mapping can be overridden by setting the **XtNmappedWhenManaged** resource for the widget when it is created or by setting the *map_when_managed* field to **False**.

To change the value of the *map_when_managed* field of a widget, use **XtSetMappedWhenManaged**. The **XtSetMappedWhenManaged** function is equivalent to calling **XtSetValues** and setting the new value for the mappedWhenManaged resource.

An alternative to using **XtSetMappedWhenManaged** to control mapping is having the client client set the *mapped_when_managed* field to **False** and calling **XtMapWidget** and **XtUnmapWidget** explicitly to map and unmap a widget.

¦ Copyright IBM Corp. 1985, 1991

4.8.6 Using Constrained Composite Widgets

Constraint widgets are a subclass of **compositeWidgetClass**. Constraint widgets manage the geometry of their children widgets based on constraints associated with each child. Constraints can be simple, such as the maximum width and height that the parent allows the child to occupy, or complex, such as instructions on how other children should change if this child is moved or resized.

Constraint widgets let a parent define resources supplied for its children. For example, if a **Constraint** parent defines the maximum sizes for its children, these new size resources are retrieved for each child as if they were resources defined by the child widget. **Constraint** resources can be included in an argument list or resource file just like any other resource for the child.

Constraint widgets have all the functionality of standard composite widgets. These widgets also process and act upon the constraint information associated with each of their children.

Every widget has a *constraints* field. This field enables widgets and the Intrinsics to keep track of the constraints associated with a child. The field contains the address of a parent-specific structure that contains constraint information about the child. If the parent is not a subclass of **constraintWidgetClass**, the *constraints* field of the child is **NULL**.

Subclasses of a **Constraint** widget can add more *constraint* fields to their superclass. To add more constraint fields, you define the constraint records in a private **.h** file using the same conventions as for widget records. For example, a widget that needs to maintain a maximum width and height for each child might define its constraint record as:

```
typedef struct {
   Dimension max_width, max_height;
} MaxConstraintPart;

typedef struct {
   MaxConstraintPart max;
} MaxConstraintRecord, *MaxConstraint;
```

A subclass of this widget that needs to maintain a minimum size can define its constraint record as:

```
typedef struct {
    Dimension min_width, min_height;
} MinConstraintPart;

typedef struct {
    MaxConstraintPart max;
    MinConstraintPart min;
} MaxMinConstraintRecord, *MaxMinConstraint;
```

Constraints are allocated, initialized, deallocated, and maintained by the Intrinsics. The constraint class record part has several entries that facilitate this control. All entries in **ConstraintClassPart** are information and procedures defined and implemented by the parent and

¦ Copyright IBM Corp. 1985, 1991

called whenever actions are performed on the children of the parent.

The **XtCreateWidget** function uses the *constraint_size* field to allocate a constraint record when a child is created. This field specifies the number of bytes occupied by a constraint record. **XtCreateWidget** also uses the constraint resources to fill in resource fields in the constraint record associated with a child. It then calls the constraint *initialize* procedure so the parent can compute constraint fields derived from constraint resources and moves or resizes the child to conform to the given constraints.

The **XtGetValues** and **XtSetValues** functions use the constraint resources to get or set the values of constraints associated with a child. **XtSetValues** then calls the constraint *set_values* procedures so the parent can recompute derived constraint fields and move or resize the child as appropriate.

The **XtDestroyWidget** function calls the constraint *destroy* procedure to deallocate dynamic storage associated with a constraint record. The constraint record must not be deallocated by the constraint destroy procedure, as **XtDestroyWidget** deallocates the constraint record automatically.

¦ Copyright IBM Corp. 1985, 1991

4.9 Using Shell Widgets

Shell widgets hold the top-level widgets of an application that communicate with the window manager. Shell widgets are designed to be as invisible as possible. A client application must create a shell widget it requires but does not handle the sizing of the widget.

If a shell widget is resized from the outside, typically by a window manager, the widget resizes its child widget automatically. Similarly, if the child widget of the shell widget wants to change size, it can make a geometry request to the shell widget to negotiate the size change with the outer environment. Client applications should not change the size of their shells directly.

Subtopics

¦ Copyright IBM Corp. 1985, 1991

4.9.1 Defining Shell Widgets

Widgets negotiate size and position with their parent widget. Widgets at the top-level of the hierarchy do not have parent widgets. Instead, these widgets must deal with the outside world. To provide for this, each top-level widget is encapsulated in a special widget called a shell widget.

Shell widgets are a subclass of the **Composite** widget. They encapsulate other widgets and allow the encapsulated widgets to avoid the geometry clipping imposed by the standard parent/child window relationship. Shell widgets also provide a layer of communication with the window manager.

There are a total of seven classes of shell widgets. Three of the classes are internal and are not instantiated or subclassed by client applications. Four of the classes are defined for public use.

The three shell widget classes allocated for internal use are:

1. **Shell**
   Provides the base class for shell widgets and the fields needed for all types of shells. **Shell** is a direct subclass of **CompositeWidgetClass**.

2. **WMShell**
   Contains fields needed by the common window manager protocol. **WMShell** is a subclass of **Shell**.

3. **VendorShell**
   Contains fields used by vendor-specific window managers. **VendorShell** is a subclass of **WMShell**.

The four shell widget classes defined for public use are:

1. **OverrideShell**
   Used for shell windows that completely bypass the window manager. A pop-up menu shell is an example of this class. **OverrideShell** is a subclass of **Shell**.

2. **TransientShell**
   Used for shell windows manipulated by the window manager but not iconified separately. These shell windows are iconified by the window manager only if the main application shell is iconified. A dialog box associated with an application shell is an example of this class. **TransientShell** is a subclass of **VendorShell**.

3. **TopLevelShell**
   Used for normal top-level windows. All additional top-level widgets required by an application are examples of this class. **TopLevelShell** is a subclass of **VendorShell**.

4. **ApplicationShell**
   Used by the window manager to define the top-level window of a separate application instance. **ApplicationShell** is a subclass of **TopLevelShell**.

¦ Copyright IBM Corp. 1985, 1991

4.9.2 Defining ShellClassPart

None of the shell widget classes has additional fields:

```
typedef struct { caddr_t extension; } ShellClassPart, OverrideShellClassPart,
     WMShellClassPart, VendorShellClassPart, TransientShellClassPart,
     TopLevelShellClassPart, ApplicationShellClassPart;
```

Shell widget classes have the (empty) shell fields immediately following the composite fields:

```
typedef struct _ShellClassRec {
     CoreClassPart core_class;
     CompositeClassPart composite_class;
     ShellClassPart shell_class;
} ShellClassRec;

typedef struct _OverrideShellClassRec {
     CoreClassPart core_class;
     CompositeClassPart composite_class;
     ShellClassPart shell_class;
     OverrideShellClassPart override_shell_class;
} OverrideShellClassRec;
typedef struct _WMShellClassRec {
     CoreClassPart core_class;
     CompositeClassPart composite_class;
     ShellClassPart shell_class;
     WMShellClassPart wm_shell_class;
} WMShellClassRec;

typedef struct _VendorShellClassRec {
     CoreClassPart core_class;
     CompositeClassPart composite_class;
     ShellClassPart shell_class;
     WMShellClassPart wm_shell_class;
     VendorShellClassPart vendor_shell_class;
} VendorShellClassRec;

typedef struct _TransientShellClassRec {
     CoreClassPart core_class;
     CompositeClassPart composite_class;
     ShellClassPart shell_class;
     WMShellClassPart wm_shell_class;
     VendorShellClassPart vendor_shell_class;
     TransientShellClassPart transient_shell_class;
} TransientShellClassRec;

typedef struct _TopLevelShellClassRec {
     CoreClassPart core_class;
     CompositeClassPart composite_class;
     ShellClassPart shell_class;
     WMShellClassPart wm_shell_class;
     VendorShellClassPart vendor_shell_class;
     TopLevelShellClassPart top_level_shell_class;
} TopLevelShellClassRec;

typedef struct _ApplicationShellClassRec {
     CoreClassPart core_class;
     CompositeClassPart composite_class;
     ShellClassPart shell_class;
```

¦ Copyright IBM Corp. 1985, 1991

```
    WMShellClassPart wm_shell_class;
    VendorShellClassPart vendor_shell_class;
    TopLevelShellClassPart top_level_shell_class;
    ApplicationShellClassPart application_shell_class;
} ApplicationShellClassRec;
```

The predefined class records and pointers for shells are:

```
extern ShellClassRec shellClassRec;
extern OverrideShellClassRec overrideShellClassRec;
extern WMShellClassRec wmShellClassRec;
extern VendorShellClassRec vendorShellClassRec;
extern TransientShellClassRec transientShellClassRec;
extern TopLevelShellClassRec topLevelShellClassRec;
extern ApplicationShellClassRec applicationShellClassRec;
extern WidgetClass shellWidgetClass;
extern WidgetClass overrideShellWidgetClass;
extern WidgetClass wmShellWidgetClass;
extern WidgetClass vendorShellWidgetClass;
extern WidgetClass transientShellWidgetClass;
extern WidgetClass topLevelShellWidgetClass;
extern WidgetClass applicationShellWidgetClass;
```

The following opaque types and opaque variables are defined for generic operations on widgets that are a subclass of **ShellWidgetClass**:

TYPES	**VARIABLES**
ShellWidget	*shellWidgetClass*
OverrideShellWidget	*overrideShellWidgetClass*
WMShellWidget	*wmShellWidgetClass*
VendorShellWidget	*vendorShellWidgetClass*
TransientShellWidget	*transientShellWidgetClass*
TopLevelShellWidget	*topLevelShellWidgetClass*
ApplicationShellWidget	*applicationShellWidgetClass*
ShellWidgetClass	
OverrideShellWidgetClass	
WMShellWidgetClass	
VendorShellWidgetClass	
TransientShellWidgetClass	
TopLevelShellWidgetClass	
ApplicationShellWidgetClass	

¦ Copyright IBM Corp. 1985, 1991

4.9.3 Defining ShellPart

The various shells have the following additional fields defined in their widget records:

```
typedef struct {
     String geometry;
     XtCreatePopupChildProc create_popup_child_proc;
     XtGrabKind grab_kind;
     Boolean spring_loaded;
     Boolean popped_up;
     Boolean allow_shell_resize;
     Boolean client_specified;
     Boolean save_under;
     Boolean override_redirect;
     XtCallbackList popup_callback;
     XtCallbackList popdown_callback;
} ShellPart;

typedef struct { int empty; } OverrideShellPart;
typedef struct {
     String title;
     int wm_timeout;
     Boolean wait_for_wm;
     Boolean transient;
     XSizeHints size_hints;
     XWMHints wm_hints;
} WMShellPart;

typedef struct {
     int vendor_specific;
} VendorShellPart;

typedef struct { int empty; }TransientShellPart;

typedef struct {
     String icon_name;
     Boolean iconic;
} TopLevelShellPart;


typedef struct {
     char *class;
     XrmClass xrm_class;
     int argc;
     char **argv;
} ApplicationShellPart;
```

The full definitions of the various shell widgets have *shell* fields following *composite* fields:

```
typedef struct {
     CorePart core;
     CompositePart composite;
     ShellPart shell;
} ShellRec, *ShellWidget;

typedef struct {
     CorePart core;
     CompositePart composite;
```

¦ Copyright IBM Corp. 1985, 1991

```
     ShellPart shell;
     OverrideShellPart override;
} OverrideShellRec, *OverrideShellWidget;

typedef struct {
     CorePart core;
     CompositePart composite;
     ShellPart shell;
     WMShellPart wm;
} WMShellRec, *WMShellWidget;
typedef struct {
     CorePart core;
     CompositePart composite;
     ShellPart shell;
     WMShellPart wm;
     VendorShellPart vendor;
} VendorShellRec, *VendorShellWidget;

typedef struct {
     CorePart core;
     CompositePart composite;
     ShellPart shell;
     WMShellPart wm;
     VendorShellPart vendor;
     TransientShellPart transient;
} TransientShellRec, *TransientShellWidget;

typedef struct {
     CorePart core;
     CompositePart composite;
     ShellPart shell;
     WMShellPart wm;
     VendorShellPart vendor;
     TopLevelShellPart topLevel;
} TopLevelShellRec, *TopLevelShellWidget;
typedef struct {
     CorePart core;
     CompositePart composite;
     ShellPart shell;
     WMShellPart wm;
     VendorShellPart vendor;
     TopLevelShellPart topLevel;
     ApplicationShellPart application;
} ApplicationShellRec, *ApplicationShellWidget;
```

¦ Copyright IBM Corp. 1985, 1991

4.9.4 Default Values for ShellPart Fields

Default values for fields common to all classes of public shells are filled in by the **Shell** resource lists and the **Shell** *initialize* procedures. The common fields and their initial default values are:

geometry
The default value for *geometry* is **NULL**. The *geometry* resource specifies size and position and is usually done only from a command line or a defaults file.

create_popup_child_proc
The default value for *create_popup_child_proc* is **NULL**. The procedure defined by this field is called by the **XtPopup** procedure.

grab_kind
The default value for *grab_kind* is (internal).

spring_loaded
The default value for *spring_loaded* is (internal).

popped_up
The default value for *popped_up* is (internal).

allow_shell_resize
The default value for *allow_shell_resize* is **False**. This field controls whether or not the widget contained by the shell is allowed to resize itself. If the value of the field is **False**, geometry requests return **XtGeometryNo**.

client_specified
The default value for *client_specified* is (internal).

save_under
The default value for *save_under* is **True** for **OverrideShell** and **TransientShell**, and **False** otherwise. Setting *save_under* instructs the server to attempt to save the contents of windows obscured by the shell when it is mapped and to restore its contents automatically later. It can be useful for pop-up menus.

override_redirect
The default value for *override_redirect* is **True** for **OverrideShell**, and **False** otherwise. Setting *override_redirect* determines whether or not the shell window is visible to the window manager. If it is **True**, the window is immediately mapped without the intervention of the manager.

popup_callback
The default value for *popup_callback* is **NULL**. This is called during **XtPopup**.

popdown_callback
The default value for *popdown_callback* is **NULL**. This is called during **XtPopdown**.

The common shell fields and their default values in **WMShell** and its subclasses are:

title
The default value for *title* is the icon name if one is specified and the name of the application otherwise. This value is a string

¦ Copyright IBM Corp. 1985, 1991

displayed by the window manager.

wm_timeout
The default value for *wm_timeout* is five seconds. This resource limits the amount of time a shell is to wait for confirmation of a geometry request to the window manager. If no confirmation comes before the defined timeout, the shell assumes the window manager is not functioning properly and sets *wait_for_wm* to **False**.

wait_for_wm
The default value for *wait_for_wm* is **True**. This field is set to **False** when a shell does not receive confirmation of a geometry request to the window manager within the time defined for *wm_timeout*. When the field is **False**, the shell does not wait for confirmation but relies on asynchronous notification.

transient
The default value for *transient* is **True** for **TransientShell**, and **False** otherwise.

min_width
The default value for *min_width* is none.

min_height
The default value for *min_height* is none.

max_width
The default value for *max_width* is none.

max_height
The default value for *max_height* is none.

width_inc
The default value for *width_inc* is none.

height_inc
The default value for *height_inc* is none.

min_aspect_x
The default value for *min_aspect_x* is none.

min_aspect_y
The default value for *min_aspect_y* is none.

max_aspect_x
The default value for *max_aspect_x* is none.

max_aspect_y
The default value for *max_aspect_y* is none.

input
The default value for *input* is **False**.

initial_state
The default value for *initial_state* is normal.

icon_pixmap
The default value for *icon_pixmap* is none.

icon_window

¦ Copyright IBM Corp. 1985, 1991

The default value for *icon_window* is none.

icon_x
The default value for *icon_x* is none.

icon_y
The default value for *icon_y* is none.

icon_mask
The default value for *icon_mask* is none.

window_group
The default value for *window_group* is none.

The common shell fields and their default values for **TopLevel** shells are:

icon_name
The default value for *icon_name* is the name of the shell widget. This field contains a string for display in the icon of the shell.

iconic
The default value for *iconic* is **False**. Setting this field to **True** is an alternative way to set the *initialState* resource to indicate that a shell is displayed initially as an icon.

The common shell fields and their default values for **Application** shells are:

argc
The default value for *argc* is 0 (zero). This field is used to initialize the standard property WM_COMMAND.

argv
The default value for *argv* is **NULL**. This field is used to initialize the standard property WM_COMMAND.

¦ Copyright IBM Corp. 1985, 1991

4.10 Using Pop-up Widgets

Pop-up widgets are used to create windows outside of the window hierarchy defined by the widget tree. A pop-up is created and attached to its widget parent differently than a standard widget child. The window associated with a pop-up child is defined as a descendant of the root window such that it is not clipped by the parent window of the pop-up widget.

A parent of a pop-up widget does not actively manage its pop-up children. A pop-up can be popped up from its parent and from other widgets.

The list of pop-up children of a parent is maintained in the *popup_list* field in the **CorePart** structure of the parent. This pop-up list defines placement in the widget hierarchy for the pop-up to get resources and provides a list of children for use by **XtDestroyWidget**. See "XtDestroyWidget" in topic 4.17.74.

A **Composite** widget can have both normal and pop-up children. The term *child* in this context refers to a standard geometry-managed child on the children list. A ***pop-up child*** refers to a child on a pop-up list.

There are three kinds of pop-up widgets:

> A ***modeless pop-up*** is usually visible to the window manager. To the user, this pop-up looks like any other application. A modeless dialog box is an example of the modeless pop-up.
>
> A ***modal pop-up*** disables user-event processing by the application except for events that occur in the pop-up. This pop-up can be visible to the window manager. An example of a modal pop-up is a modal dialog box.
>
> A ***spring-loaded pop-up*** disables user-event processing by all applications except for events that occur in the pop-up. This pop-up is not visible to the window manager. An example of a spring-loaded pop-up is a menu.

Modal pop-ups and spring-loaded pop-ups are similar and should be coded the same. A single widget, such as a **ButtonBox** or a **Menu**, can be used as a modal pop-up and as a spring-loaded pop-up within the same application. The main difference between the modal pop-up and the spring-loaded pop-up is that each spring-loaded pop-up is brought up with the pointer and requires different processing by the Intrinsics because of the grab caused by the pointer button.

Any kind of pop-up can pop up other widgets. Modal and spring-loaded pop-ups can constrain user events to the most recent pop-up or to any modal or spring-loaded pop-ups currently mapped.

Pop-up widget classes are subclasses of **Shell** responsible for communicating with the window manager.

Subtopics

¦ Copyright IBM Corp. 1985, 1991

4.10.1 Creating a Pop-up Shell

For a widget to pop up, it must be the child of a pop-up widget shell. The ***pop-up shell*** communicates with the window manager when geometry requests are made. It is also responsible for handling the bookkeeping associated with actual pop-up and pop-down actions. Each pop-up shell can have only one pop-up child. The shell and child together are referred to as the ***pop-up***. When a child pop-up is required, the pop-up shell must be specified.

To create a pop-up shell, you use **XtCreatePopupShell**. This function attaches a pop-up shell to the pop-up list of the parent. See "XtCreatePopupShell" in topic 4.17.68.

A spring-loaded pop-up invoked from a translation table must already exist at the time the transaction is invoked so the translation manager can locate the shell by name. Other pop-ups can be created as needed. Creating a pop-up later can be useful when you pop up an unspecified number of pop-ups. You can determine if an unused shell or a shell not currently popped up exists, and then create a new shell if necessary.

¦ Copyright IBM Corp. 1985, 1991

4.10.2 Creating Pop-up Children

The child of a pop-up shell can be created anytime after the pop-up shell is created. The pop-up child can be treated as either a static and dynamic entity.

An application can create the child of the pop-up shell at application startup. This action is appropriate for pop-up children composed of a fixed set of widgets. The application can change the state of the subparts of the pop-up child as the application state changes. For example, if an application creates a static menu, it can call **XtSetSensitive** or **XtSetValues** on any of the buttons that make up the menu. Creating the pop-up child at application startup means that pop-up time can be minimized, especially if the application calls **XtRealizeWidget** on the pop-up shell at startup time. When the pop-up child is needed, the widgets that make up the child already exist and only need to be mapped.

Alternatively, an application can postpone the creation of a pop-up child until it is needed. This minimizes application startup time and allows the pop-up child to reconfigure itself dynamically each time it is popped up. In this case, the pop-up child creation routine polls the application to find out if it should change any of its subparts.

Creating a pop-up child does not map the pop-up.

All shells have pop-up and pop-down callbacks. These callbacks provide mechanisms to make last-minute changes to a pop-up child before it is popped up or to change the child after it is popped down. Excessive use of pop-up callbacks can slow the pop-up action.

¦ Copyright IBM Corp. 1985, 1991

4.10.3 Mapping a Pop-up Widget

Pop-ups can be popped up and mapped through several mechanisms:

- A call to **XtPopup**
- Use of a supplied callback procedure such as **XtCallbackNone**, **XtCallbackNonexclusive**, or **XtCallbackExclusive**
- Use of the standard translation action **MenuPopup**.

A call to **XtPopup** maps a pop-up from within an application. The procedures **XtCallbackNone**, **XtCallbackNonexclusive**, and **XtCallbackExclusive** map a pop-up from the callback list of a specified widget. **MenuPopup** pops up a menu when a pointer button is pressed or when the pointer is moved into a window defined to produce the menu.

¦ Copyright IBM Corp. 1985, 1991

4.10.4 Unmapping a Pop-up Widget

Pop-ups can be popped down and unmapped through several mechanisms:

A call to **XtPopdown**

Use of the supplied callback procedure **XtCallbackPopdown**

Use of the standard translation action **MenuPopdown**.

A call to **XtPopdown** pops down and unmaps a pop-up from within an application. The procedure **XtCallbackPopdown** pops down and unmaps pop-ups popped up by **XtCallbackNone**, **XtCallbackNonexclusive**, and **XtCallbackExclusive**. **MenuPopdown** pops down and unmaps a spring-loaded menu when a pointer button is released or when the pointer is moved into a window.

¦ Copyright IBM Corp. 1985, 1991

4.11 Using the Utility Functions

The Toolkit provides a number of utility functions for:

- Memory managemen
- Graphics Contexts sharin
- Exposure region
- Error handling

Subtopics

¦ Copyright IBM Corp. 1985, 1991

4.11.1 Managing Memory

The Toolkit memory management routines provide uniform checking for null pointers and error reporting on memory allocation errors. These routines are completely compatible with the standard C language run time routines **malloc**, **calloc**, **realloc**, and **free** with the added functionality:

XtMalloc, **XtNew**, **XtCalloc**, and **XtRealloc** return an error if there is insufficient memory.
XtFree returns if a null pointer is passed.
XtRealloc allocates new storage if a null pointer is passed.

See "Allocating and Deallocating Memory" in topic 6.9.

¦ Copyright IBM Corp. 1985, 1991

4.11.2 Sharing Graphics Contexts

The Toolkit provides a mechanism that allows clients to share **Graphics Contexts** (GCs), reducing both the number of GCs created and the number of calls to the server in an application. **XtGetGC** obtains shared **GCs**. The **GCs** must be read-only when using **XtGetGC**. If a changeable **GC** is needed, use the **Xlib XCreateGC** function instead. See Chapter 2, "X-Windows Xlib Functions" in topic 2.0.

¦ Copyright IBM Corp. 1985, 1991

4.11.3 Merging Exposure Events into a Region

The Toolkit provides **XtAddExposureToRegion** that merges **Expose** and **GraphicsExpose** events into a region that clients can process at once, rather than processing individually. See Chapter 1, "Using X-Windows" in topic 1.0.

¦ Copyright IBM Corp. 1985, 1991

4.11.4 Handling Errors

The Toolkit allows a client to register a procedure which is called when an error occurs. This facility reports and logs errors, but does not correct them or help you recover.

¦ Copyright IBM Corp. 1985, 1991

4.12 Managing Events

While X-Windows allows the reading and processing of events anywhere in an application, widgets in the Toolkit can neither directly read events nor grab the server or pointer. Widgets register procedures to call when an event or class of events occurs in that widget.

A typical application consists of start-up code followed by an event loop that reads events and dispatches them by calling the procedures that widgets have registered. The default event loop provided by the Intrinsics is **XtAppMainLoop**. See "XtAppMainLoop" in topic 4.17.29.

The event manager is a collection of functions that:

Add or remove event sources other than X server events, particularl timer interrupts and file input

Query the status of event source

Add or remove procedures to be called when an event occurs for particular widget

Enable and disable the dispatching of user-initiated events, such a keyboard and pointer events, for a particular widget

Constrain the dispatching of event to a cascade of pop-up widget

Call the appropriate set of procedures currently registered when a event is read.

Most widgets do not need to call the event handler functions explicitly. The normal interface to X events is through the higher-level translation manager. The translation manager maps sequences of X events, with modifiers, into procedure calls. Applications rarely use event manager routines other than **XtAppMainLoop**.

Subtopics

¦ Copyright IBM Corp. 1985, 1991

4.12.1 Adding and Deleting Additional Event Sources

While most applications are driven only by X events, some applications need to incorporate other sources of input into the X Toolkit event handling mechanism. The event manager provides routines to integrate notification of timer events and file data pending into this mechanism.

Subtopics
4.12.1.1 Adding and Removing Input Sources
4.12.1.2 Adding and Removing Timeouts

¦ Copyright IBM Corp. 1985, 1991

4.12.1.1 Adding and Removing Input Sources

Two functions, **XtAppAddInput** and **XtRemoveInput**, control input gathering from files. The application registers the files with the Intrinsics *read* routine. When input is pending on one of the files, the registered callback procedures are invoked.

XtAppAddInput registers a new file as an input source for a given application. See "XtAppAddInput" in topic 4.17.20.

XtRemoveInput discontinues a source of input. See "XtRemoveInput" in topic 4.17.151.

¦ Copyright IBM Corp. 1985, 1991

4.12.1.2 Adding and Removing Timeouts

The timeout facility notifies the application or the widget through a callback procedure that a specified time interval has elapsed. Timeout values are uniquely identified by an interval ID.

XtAppAddTimeOut creates a timeout value. See "XtAppAddTimeOut" in topic 4.17.21.

XtRemoveTimeOut clears a timeout value. See "XtRemoveTimeOut" in topic 4.17.153.

¦ Copyright IBM Corp. 1985, 1991

4.12.2 Compressing Events Using Event Filters

The event manager provides filters that can be applied to X user events. The filters screen out events that are redundant or temporarily unwanted. These filters handle:

- Pointer motion compressio
- Enter/leave compressio
- Exposure compression

Subtopics

¦ Copyright IBM Corp. 1985, 1991

4.12.2.1 Pointer Motion Compression

Widgets can have a hard time keeping up with pointer motion events and rarely need notification of every motion event. Setting the widget class field *compress_motion* to **True** filters out redundant motion events. With this setting, when a request for an event returns a motion event, the Intrinsics checks for other motion events immediately following the current one and ignores all but the last of them.

¦ Copyright IBM Corp. 1985, 1991

4.12.2.2 Enter/Leave Compression

Sometimes pairs of enter and leave events with no intervening events can be ignored. An example is when a user moves the pointer across a widget without stopping in it. Setting the widget class field *compress_enterleave* to **True** filters out pairs of enter and leave events with no intervening events. With this setting, enter and leave events are not delivered to the client if they are found together in the input queue.

¦ Copyright IBM Corp. 1985, 1991

4.12.2.3 Exposure Compression

Many widgets prefer to process a series of exposure events as a single expose region rather than as individual rectangles. Widgets with complex displays might use the expose region as a clip list in a graphics context. Widgets with simple displays might ignore the region entirely and redisplay their whole window or get the bounding box from the region and redisplay only that rectangle.

In either case, these widgets do not require information on partial expose events. If the *compress_exposure* field in the widget class structure is set to **True**, the event manager calls the widget's *expose* procedure only once for each series of exposure events. In this case, all **Expose** events are accumulated into a region. When the final **Expose** event in a series (one with count zero) is received, the event manager replaces the rectangle in the event with the bounding box for the region and calls the widget's *expose* procedure, passing the modified exposure event and the region.

If *compress_exposure* is **False**, the event manager calls the widget's *expose* procedure for every exposure event, passing it the event and a region argument of **NULL**.

¦ Copyright IBM Corp. 1985, 1991

4.12.3 Constraining Events

Modal widget pop-ups lock out user input to an application except direct input to that widget. (See "Using Pop-up Widgets" in topic 4.10.)

When a modal menu or modal dialog box is popped up using **XtPopup**, user events such as keyboard and pointer events that occur outside the modal widget are delivered to the modal widget or ignored. In this case, user events are not delivered to a widget outside of the modal widget.

Menus can pop up submenus and dialog boxes can pop up further dialog boxes to create a ***pop-up cascade***. In this case, user events are delivered to one of several modal widgets in the cascade.

Display-related events are delivered outside the modal cascade so that expose events keep the application display current. Events that occur within the cascade are delivered normally. User events that are delivered to the most recent spring-loaded shell in the cascade when they occur outside the cascade are called ***remap events***. These remap events include **KeyPress**, **KeyRelease**, **ButtonPress**, and **ButtonRelease** events. **MotionNotify**, **EnterNotify**, and **LeaveNotify** events are ignored if they occur outside the cascade. All other events are delivered normally.

XtPopup uses the **XtAddGrab** and **XtRemoveGrab** functions to constrain user events to a modal cascade and subsequently to remove a grab when the modal widget goes away. There is usually no need to call them explicitly.

The modal cascade is used by **XtDispatchEvent** when it tries to dispatch a user event. When at least one modal widget is in the widget cascade, **XtDispatchEvent** first determines if the event should be delivered. It starts at the most recent cascade entry and follows the cascade up to and including the most recent cascade entry added with the exclusive parameter **True**.

This subset of the modal cascade along with all descendants of these widgets is the active subset. User events that occur outside the widgets in this subset are ignored or remapped. Modal menus with submenus generally add a submenu widget to the cascade with exclusive **False**. Modal dialog boxes that must restrict user input to the most deeply nested dialog box add a subdialog widget to the cascade with exclusive **True**. User events that occur within the active subset are delivered to the appropriate widget, usually a child or further descendant of the modal widget.

Regardless of where on the screen they occur, remap events are always delivered to the most recent widget in the active subset of the cascade with its *spring_loaded* field set to **True**, if any such widget exists.

XtAddGrab redirects user input to a modal widget.

XtRemoveGrab removes the redirection of user input to a modal widget.

¦ Copyright IBM Corp. 1985, 1991

4.12.4 Focusing Events on a Child

XtSetKeyboardFocus redirects keyboard input to a child of a **Composite** widget without calling **XSetInputFocus**. Widgets requiring the input focus can call **XSetInputFocus** explicitly.

To allow outside agents to cause a widget to get the input focus, each widget exports an *accept_focus* procedure. The widget returns whether it actually took the focus or not, so that the parent can give the focus to another widget.

Widgets that need to know when they lose the input focus must use the **Xlib** focus notification mechanism explicitly by specifying translations for **FocusIn** and **FocusOut** events. Widgets not requiring information on input focus have their *accept_focus* procedure pointer set to **NULL**.

XtCallAcceptFocus calls the *accept_focus* procedure of a widget.

¦ Copyright IBM Corp. 1985, 1991

4.12.5 Querying Event Sources

The event manager provides three functions to examine and read events, including file and timer events, that are in the queue. These functions handle Intrinsics equivalents of the **Xlib** calls **XPending**, **XPeekEvent**, and **XNextEvent**.

XtAppPending determines if there are events on the input queue for a given application.

XtAppPeekEvent returns the value from the beginning of the input queue of a given application without removing input from the queue.

XtAppNextEvent returns the value from the beginning of the input queue of a given application.

¦ Copyright IBM Corp. 1985, 1991

4.12.6 Dispatching Events

The Intrinsics provide functions that dispatch events to widgets or other application code. Every client interested in X events on a widget uses **XtAddEventHandler** to register which events it is interested in and a procedure (event handler) to be called when the event happens in that window. The translation manager automatically registers event handlers for widgets that use translation tables.

XtAppProcessEvent gives applications direct control of the processing of different types of input. This procedure is not usually called by client applications. **XtAppProcessEvent** processes timer events by calling appropriate timer callbacks, alternate input by calling appropriate alternate input callbacks, and X events by calling **XtDispatchEvent**.

¦ Copyright IBM Corp. 1985, 1991

4.12.7 Handling the Application Input Loop

XtAppMainLoop handles the processing of input from a given application. This procedure controls the main loop of X Toolkit applications and does not return. This main loop is basically an infinite loop that calls **XtAppNextEvent** and then **XtDispatchEvent**. Applications exit in response to some user action.

Applications can provide their own version of this loop by testing a global termination flag or confirming that the number of top-level widgets is larger than zero before circling back to the call to **XtAppNextEvent**.

¦ Copyright IBM Corp. 1985, 1991

4.12.8 Setting and Checking the Sensitivity State of a Widget

Many widgets have a mode in which they assume a different appearance (for example, are greyed out or stippled), do not respond to user events, and become dormant. When dormant, a widget is considered to be ***insensitive***. When a widget is insensitive, the event manager does not dispatch events to the widget with an event type of **KeyPress**, **KeyRelease**, **ButtonPress**, **ButtonRelease**, **MotionNotify**, **EnterNotify**, **LeaveNotify**, **FocusIn**, or **FocusOut**.

A widget can be insensitive because its *sensitive* field is set to **False** or because one of its parents is insensitive, causing the widget's *ancestor_sensitive* field to be set to **False**. A widget can but does not need to distinguish these two cases visually.

XtSetSensitive sets the sensitivity state of a widget.

XtIsSensitive checks the current sensitivity satte of a given widget. This procedure is usually done by parents.

¦ Copyright IBM Corp. 1985, 1991

4.12.9 Adding Background Work Procedures

The Intrinsics have limited support for background processing. Because most applications spend most of their time waiting for input, you can register an idle-time work procedure to be called when the Toolkit would otherwise blick in **XtAppNextEvent** or **XtAppProcessEvent**. Work procedure pointers are of type **XtWorkProc**.

XtAppAddWorkProc registers a work procedure for a given application. Multiple work procedures can be registered.

To remove a work procedure, either return **True** from the procedure when it is called or use **XtRemoveWorkProc**.

¦ Copyright IBM Corp. 1985, 1991

4.12.10 Handling Widget Exposure and Visibility

Every primitive widget and some composite widgets display data on the screen by means of raw **Xlib** calls. Widgets must keep track of what they write to the screen. They must keep enough state to redisplay the window or parts of it if a portion is obscured and then redisplayed.

Subtopics
4.12.10.1 Redisplaying a Widget
4.12.10.2 Widget Visibility

¦ Copyright IBM Corp. 1985, 1991

4.12.10.1 Redisplaying a Widget

The redisplay of a widget upon exposure is the responsibility of the *expose* procedure in the class record of the widget. The *expose* procedure pointer in a widget class is of type **XtExposeProc**.

If a widget has no display semantics, it can specify **NULL** for the *expose* field. If the *expose* procedure is **NULL**, **XtRealizeWidget** fills in a default bit gravity of **NorthWestGravity** before it calls the *realize* procedure of the widget.

Many composite widgets serve only as containers for their children and have no defined *expose* procedure.

It often is possible to anticipate the display needs of several levels of subclassing. For example, rather than writing separate display procedures for the widgets **Label**, **Command**, and **Toggle**, you can write a single display routine in **Label** that uses display state fields like the following:

Boolean *invert*
Boolean *highlight*
Dimension *highlight_width*

Label would have *invert* and *highlight* always **False** and *highlight_width* zero. **Command** would dynamically set *highlight* and *highlight_width*, but it would leave *invert* always **False**. **Toggle** would dynamically set all three. In this case, the *expose* procedures for **Command** and **Toggle** inherit the *expose* procedure of their superclass.

¦ Copyright IBM Corp. 1985, 1991

4.12.10.2 Widget Visibility

Some widgets use substantial computing resources to display data. However, this resource is wasted if the widget is not actually visible on the screen, obscured by another application, or iconified.

The *visible* field in the **Core** widget structure tells the widget that it need not display data. This field is **True** by the time an **Expose** event is processed if the widget is visible, but is usually **False** if the widget is not visible.

Widgets can use or ignore the *visible* field. They ignore the field if *visible_interest* in their widget class record is set to **False**. In such cases, the *visible* field is initialized **True** and does not change. If *visible_interest* is **True**, the event manager asks for **VisibilityNotify** events for the widget and updates the *visible* field accordingly.

¦ Copyright IBM Corp. 1985, 1991

4.12.11 Using X Event Handlers

Event handlers are procedures called when specified events occur in a widget. Most widgets do not use event handlers explicitly. Instead, they use the Intrinsics translation manager.

Event handler procedure pointers are of type **XtEventHandler**.

Subtopics
4.12.11.1 Event Handlers That Select Events
4.12.11.2 Event Handlers That Do Not Select Events
4.12.11.3 Retrieving the Current Event Mask

¦ Copyright IBM Corp. 1985, 1991

4.12.11.1 Event Handlers That Select Events

XtAddEventHandler registers an event handler procedure with the dispatch mechanism. This procedure is then called when an event matching the defined mask occurs on the specified widget.

XtRemoveEventHandler removes a previously registered event handler.

¦ Copyright IBM Corp. 1985, 1991

4.12.11.2 Event Handlers That Do Not Select Events

XtAddRawEventHandler enables clients to register an event handler procedure with the dispatch mechanism without causing the server to select for that event. It functions like **XtAddEventHandler** except that it does not affect the mask of the widget and does not cause an **XSelectInput** for its events.

XtRemoveRawEventHandler removes a previously registered raw event handler.

¦ Copyright IBM Corp. 1985, 1991

4.12.11.3 Retrieving the Current Event Mask

XtBuildEventMask retrieves the event mask for a given widget.

¦ Copyright IBM Corp. 1985, 1991

4.13 Managing Widget Geometry

A widget does not directly control its size or location. The position of children is usually the responsibility of the parent. However, the child widgets often have the best idea of their optimal sizes and preferred locations.

To resolve physical layout conflicts between sibling widgets or a widget and its parent, the Intrinsics provide a geometry management mechanism. Most **Composite** widgets have a *geometry_manager* field in the widget class record that is responsible for the size, position, and stacking order of the widget's children. Exceptions to this are fixed boxes which create children and can ensure that these children never initiate a geometry request.

Subtopics

¦ Copyright IBM Corp. 1985, 1991

4.13.1 Initiating Geometry Changes

Parents, children, and clients initiate geometry changes differently. Because a parent has absolute control of the geometry of its children, it changes the geometry directly by calling **XtMoveWidget**, **XtResizeWidget**, or **XtConfigureWidget**. A child must ask its parent for a geometry change by calling **XtMakeGeometryRequest** or **XtMakeResizeRequest** to convey its request to its parent. An application or other client code initiates a geometry change by calling **XtSetValues** on the appropriate geometry fields, thereby giving the widget the opportunity to modify or reject the client request before it gets propagated to the parent and the opportunity to respond appropriately to the reply of the parent.

When the geometry manager of a parent received a request from a child widget for a change in the child's size, position, border width, or stacking depth, the geometry manager can:

Allow the reques
Disallow the reques
Suggest a compromise

When the geometry manager is asked to change the geometry of a child, the geometry manager may also rearrange and resize any or all of the other children that it controls. The geometry manager can move children around freely using **XtMoveWidget**. When it resizes a child other than the one making the request, it calls **XtResizeWidget**. The geometry manager can simultaneously move and resize a child with a single call to **XtConfigureWidget**.

Often, geometry managers find they can satisfy a request only by reconfiguring a widget they do not control, such as when the **Composite** widget wants to change its own size. In this case, the geometry manager makes a request to the geometry manager of the parent. Geometry requests can cascade this way to arbitrary depth.

Because such cascaded arbitration of widget geometry can involve extended negotiation, windows are not actually allocated to widgets at application startup until all widgets are satisfied with their geometry. For more information, see "XtRealizeWidget" in topic 4.17.142 and "Creating Widgets" in topic 4.7.

Notes:

1. After a successful geometry request (one that returns **XtGeometryYes**), a widget does not know whether or not its *resize* procedure has been called. Widgets should have *resize* procedures that can be called more than once without ill effects.

2. The Intrinsics treatment of stacking requests is deficient in several areas. Stacking requests for unrealized widgets are granted but will have no effect. In addition, there is no way to do an **XtSetValues** to generate a stacking geometry request.

¦ Copyright IBM Corp. 1985, 1991

4.13.2 Making General Geometry Manager Requests

To make a general geometry manager request from a widget, you use **XtMakeGeometryRequest**.

The return codes form geometry managers are:

```
typedef enum _XtGeometryResult {
        XtGeometryYes,
        XtGeometryNo,
        XtGeometryAlmost,
        XtGeometryDone,
} XtGeometryResult;
```

The **XtWidgetGeometry** structure is similar to a corresponding **Xlib** structure:

```
typedef unsigned long XtGeometryMask;
typedef struct {
        XtGeometryMask request_mode;
        Position x, y;
        Dimension width, height;
        Dimension border_width;
        Widget sibling;
        int stack_mode;
} XtWidgetGeometry;
```

The following *request_mode* values are from **X.h**:

```
#define CWX                 (1<<0)

#define CWY                 (1<<1)

#define CWWidth             (1<<2)

#define CWHeight            (1<<3)

#define CWBorderWidth       (1<<4)

#define CWSibling           (1<<5)

#define CWStackMode         (1<<6)
```

The Intrinsics also support the following value:

```
#define XtCWQueryOnly       (1<<7)
```

XtCWQueryOnly indicates that the corresponding geometry request is only a query asking what would happen if this geometry request were made and that no widgets are actually changed.

XtMakeGeometry (like the corresponding **XConfigureWindow Xlib** function) uses *request_mode* to determine which fields in the **XtWidgetGeometry** structure you want to specify.

The *stack_mode* values are defined in **X.h**:

```
#define Above               0

#define Below               1
```

¦ Copyright IBM Corp. 1985, 1991

```
#define TopIf                   2

#define BottomIf                3

#define Opposite                4
```

The Intrinsics also support the following value:

```
#define XtSMDontChange          5
```

XtSMDontChange indicates that the widget requires its current stacking order preserved.

¦ Copyright IBM Corp. 1985, 1991

4.13.3 Making Resize Requests

To make a simple resize request from a widget, you can use **XtMakeResizeRequest**. This function is basically a simple interface to **XtMakeGeometryRequest**.

¦ Copyright IBM Corp. 1985, 1991

4.13.4 Managing Potential Geometry Changes

Sometimes a geometry manager cannot respond to a geometry request from a child without first making a geometry request to the widget's own parent (the requestor's grandparent). If the request to the grandparent would allow the parent to satisfy the original request, the geometry manager can make the intermediate geometry request as if it were the originator.

On the other hand, if the geometry manager already has determined that the original request cannot be completely satisfied (for example, if it always denies position changes), it tells the grandparent to respond to the intermediate request without actually changing the geometry because it does not know if the child will accept the compromise. To accomplish this, the geometry manager uses **XtCWQueryOnly** in the intermediate request.

When **XtCWQueryOnly** is used, the geometry manager needs to cache enough information to exactly reconstruct the intermediate request. If the grandparent's response to the intermediate query is **XtGeometryAlmost**, the geometry manager caches the entire reply geometry in the event the child accepts the parent's compromise.

If the grandparent's response is **XtGeometryAlmost**, the entire reply geometry from the grandparent is cached when **XtCWQueryOnly** is not used. If the geometry manager is still able to satisfy the original request, it immediately accepts the grandparent's compromise and acts on the child's request. If the grandparent's compromise geometry is insufficient to allow the child's request and if the geometry manager is willing to offer a different compromise to the child, the grandparent's compromise is not accepted until the child has accepted the new compromise.

A compromise geometry returned with **XtGeometryAlmost** is guaranteed only for the next call to the same widget. Therefore a cache of size one is sufficient.

¦ Copyright IBM Corp. 1985, 1991

4.13.5 Managing Child Geometry

The *geometry_manager* procedure manages the geometry of a child. The procedure can change the *x*, *y*, *width*, *height*, and *border_width* values of a widget. The *geometry_manager* procedure pointer in a composite widget class is of type **XtGeometryHandler**.

Sometimes the geometry manager can satisfy change requests only in part or in some altered form. For instance, the manager may fill a subset of requests such as size or position, or a request in a modified form, such as resizing a widget's window but not exactly to the requested size.

In such cases, changes are made only if the child agrees to the compromises. The geometry manager fills in *geometry_return* with the actual changes it can make, including an appropriate mask, and returns **XtGeometryAlmost**. If a bit in *geometry_return->request_mode* is zero, the geometry manager does not change the corresponding value is the *geometry_return* is used immediately in a new request. If a bit is one, the geometry manager does change that element to the corresponding value in *geometry_return*. More bits may be set in *geometry_return* than in the original request if the geometry manager in set to change other fields upon compromise acceptance from the child.

When **XtGeometryAlmost** is returned, the widget must decide if the compromise suggested in *geometry_return* is acceptable. If it is, the widget makes a call to **XtMakeGeometryRequest** to change its geometry.

The geometry manager must grant the request if the next geometry request from this child uses the *geometry_return* box filled in by an **XtGeometryAlmost** return, if there have been no intervening geometry requests on either its parent or any of its other children, and if possible. If the child gives its comfirmation immediately with the returned geometry, it usually gets an answer of **XtGeometryYes**. However, the user's window manager can affect the final outcome.

To return an **XtGeometryYes**, the geometry manager frequently rearranges the position of other managed children by calling **XtMoveWidget**. However, some geometry managers change the size of other managed children by calling **XtResizeWidget** or **XtConfigureWidget**. If **XtCWQueryOnly** is specified, the geometry manager must return how it will react to this geometry request without actually moving or resizing any widgets.

Geometry managers do not assume that the *request* and *geometry_return* arguments point to independent storage. The caller is permitted to use the same field for both. The geometry manager allocates its own temporary storage if necessary.

¦ Copyright IBM Corp. 1985, 1991

4.13.6 Managing Widget Placement and Size

A child can be resized by its parent at any time. Widgets usually need to know when they have changed size so that they can layout their displayed data to match the new size.

If a class need not recalculate anything when a widget is resized, it can specify **NULL** for the *resize* field in its class record. This case occurs only for widgets with simple display semantics. The *resize* procedure takes a widget as its only argument. The *x*, *y*, *width*, *height*, and *border_width* fields of the widget contain the new values. The *resize* procedure recalculates the layout of internal data as required. For example, a centered label in a window that changes size recalculates the starting position of the text.

The widget must make changes requested by the *resize* procedure. A widget must not issue an **XtMakeGeometryRequest** or **XtMakeResizeRequest** call from its *resize* procedure.

XtResizeWidget is used to resize a sibling widget of a child making a geometry request. This procedure updates the geometry fields in the widget, configures the window if the widget is realized, and calls the *resize* procedure of the child to notify it of the changes. The *resize* procedure pointer is of type **XtWidgetProc**.

XtResizeWindow is used to resize a child widget that already has the new values of its width, height, and border width fields.

XtConfigureWidget is used to move and resize a sibling widget of a child making a geometry request.

XtMoveWidget is used to move a sibling widget of a child making a geometry request.

¦ Copyright IBM Corp. 1985, 1991

4.13.7 Handling Preferred Geometry

Parents can sometimes adjust their layouts to accommodate the preferred geometries of their children. They can use **XtQueryGeometry** to obtain the preferred geometry and use or ignore any portion of the response. Parents are expected to call **XtQueryGeometry** in their layout routine and wherever other information is significant after a *change_managed* procedure is called.

¦ Copyright IBM Corp. 1985, 1991

4.14 Resource Management

Widget writers need a large set of resources at the time of widget creation. Some resources come from the resource database, some from the argument list supplied in the call to **XtCreateWidget**, and some from the internal defaults specified for the widget. Resources are obtained first from the argument list, then from the resource database for all resources not specified in the argument list, and last from the internal default.

A resource is a field in the widget record with a corresponding resource entry in the widget resource list or in the resource list of any of its superclasses. This field can be set by:

XtCreateWidget, by naming the field in the argument list.

an entry in the default resource files, using either the name o class.

using **XtSetValues**.

In addition, this field is readable by **XtGetValues**.

Not all fields in a widget record are resources. Some fields provide bookkeeping for the generic routines, such as *managed* and *being_destroyed*. Other fields provide local bookkeeping while still others are derivations of resources, such as **GCs** and Pixmaps.

Subtopics
4.14.1 Creating Resource Lists
4.14.2 Chaining Resource Lists from Superclass to Subclass
4.14.3 Converting Resources

¦ Copyright IBM Corp. 1985, 1991

4.14.1 Creating Resource Lists

A resource entry specifies a field in the widget; the textual name and class of the field that argument lists and external resource files use to refer to the field; and a default value that the field should get if no value is specified.

The declaration for the **XtResource** structure is:

```
typedef struct {
     String resource_name;
     String resource_class;
     String resource_type;
     Cardinal resource_size;
     Cardinal resource_offset;
     String default_type;
     caddr_t default_address;
} XtResource, *XtResourceList;
```

The following list describes the content of each of these structure fields.

The *resource_name* contains the name used by clients to access the field in the widget. This names starts with a lower-case letter. It is spelled almost the same as the field name, except that each underbar character is replaced by an uppercase letter. For example, the resource name for *background_pixel* is **backgroundPixel**. Widget header files typically contain a symbolic name for each resource name. All resource names, classes, and types used by the Intrinsics are in the file **<X11/StringDefs.h>**. The Intrinsics symbolic resource names begin with **XtN** and are followed by the string name, for example, **XtNbackgroundPixel** for **backgroundPixel**.

A *resource class* has two functions:

- It isolates an application from different representations that widgets can use for a similar resource.

- It lets an application specify values for several resources with a single name. A resource class should be chosen to span a group of closely related fields.

For example, a widget can have several pixel resources, such as *background, foreground, border, block cursor, mouse cursor,* and so on. Typically, the *background* defaults to *white* and everything else defaults to *black*. The resource class for each of these resources in the resource list should be chosen so that it takes a minimal number of entries in the resource database to make *background offwhite* and everything else *darkblue*.

In this case, the background pixel should have a resource class of **Background** and all the other pixel entries should have a resource class of **Foreground**. Then, the resource file needs just two lines to change all pixels to *offwhite* or *darkblue*:

```
Background:     offwhite
Foreground:     darkblue
```

Similarly, a widget may have several resource fonts, such as normal

¦ Copyright IBM Corp. 1985, 1991

and bold, but all fonts should have the class **Font**. To change all fonts requires one line in the default file:

```
Font: Rom14.500
```

Resource class names begin with a capitalized letter. This name is preceded by **XtC**, such as **XtCBackground**.

The *resource_type* field is the physical representation type of the resource. This name begins with an uppercase letter and is spelled the same as the type name of the field. The resource type is used when resources are called to convert from the resource database format (usually String) or the default resource format (often String) to the desired physical representation. For more information, see "Converting Resources" in topic 4.14.3. The Intrinsics define the following resource types:

Resource Type	Structure or Field Type
XtRAcceleratorTable	XtAccelerators
XtRBoolean	Bool
XtRBool	Bool
XtRCallback	XtCallbackList
XtRColor	XColor
XtRCursor	Cursor
XtRDimension	Dimension
XtRDisplay	Display*
XtRFile	FILE*
XtRFloat	float
Resource Type	**Structure or Field Type**
XtRFont	Font
XtRFontStruct	XFontStruct*
XtRFunction	(*)()
XtRInt	int
XtRPixel	Pixel
XtRPixmap	Pixmap
XtRPointer	caddr_t
XtRPosition	Position
XtRShort	short

¦ Copyright IBM Corp. 1985, 1991

XtRString	char*
XtRTranslationTable	XtTranslations
XtRUnsignedChar	unsigned char
XtRWidget	Widget
XtRWindow	Window

The *resource_size* field is the size of the physical representation in bytes and should be specified as **sizeof(***type***)** so that the compiler can fill in the value.

The *resource_offset* is the offset in bytes within the widget. Use **XtOffset** to retrieve this value.

The *default_type* is the representation type of the default resource value. If *default_type* is different from *resource_type* and the *default_type* is needed, the resource manager invokes a conversion procedure from *default_type* to *resource_type*. Whenever possible, the default type should be identical to the resource type to minimize widget creation time. However, there are sometimes no values of the type that the application can easily specify. In this case, the value should be one that the converter will work for, such as **XtDefaultForeground** for a pixel resource.

The *default_address* is the address of the default resource value. The default is used if a resource is not specified in the argument list or in the resource database or if the conversion from the representation type stored in the resource database fails, which can happen for reasons such as a misspelled entry in a resource file.

Two special representation types, **XtRImmediate** and **XtRCallProc**, can only be used as default resource types. **XtRImmediate** indicates that the value in the *default_address* field is the actual value of the resource, rather than the address of the value. The value must be in correct representation type for the resource. No conversion is possible since there is no source representation type.

XtRCallProc indicates that the value in the *default_address* field is a procedure variable. This procedure is automatically invoked with the *widget*, *resource_offset*, and a pointer to the **XrmValue** in which to store the result. The procedure is of type **XtResourceDefaultProc**.

XtGetResourceList gets the resource list structure for a particular class. **XtSetValues** and **XtGetValues** use the resource list to set and get widget state. For further information, see "XtGetValues" in topic 4.17.98, "XtGetSubvalues" in topic 4.17.97, and "XtSetValues" in topic 4.17.169.

An abbreviated version of the resource list in the Label widget:

Resources specific to Label:

```
static XtResource resources[ ] = {
   {XtNforeground, XtCForeground, XtRPixel, sizeof(Pixel),
      XtOffset(LabelWidget, label.foreground), XtRString, XtDefaultForeground
   {XtNfont, XtCFont, XtRFontStruct, sizeof(XFontStruct *),
       XtOffset(LabelWidget, label.font),XtRString, XtDefaultFont},
```

¦ Copyright IBM Corp. 1985, 1991

```
   {XtNlabel,  XtCLabel, XtRString, sizeof(String),
      XtOffset(LabelWidget, label.label), XtRString, NULL},
                          .
                          .
                          .
}
```

The complete resource name for a field of a widget instance is the concatenation of the application name (from **XtAppCreateShell**), the instance names of all the parents of the widget up to the **ApplicationShellWidget**, the instance name of the widget itself, and the resource name of the specified field of the widget.

The full resource class of a field of a widget instance is the concatenation of the application class (from **XtAppCreateShell**), the widget class names of all the parents of the widget up to the **ApplicationShellWidget** (not the superclasses), the widget class name of the widget itself, and the resource name of the specified field of the widget.

¦ Copyright IBM Corp. 1985, 1991

4.14.2 Chaining Resource Lists from Superclass to Subclass

The procedure **XtCreateWidget** gets resources as a superclass-to-subclass operation. The resources specified in the the Core resource list are called, then those resources in the subclass are called, and so on down to the resources specified for this widget class. Within a class, resources are called in the order they are declared.

Generally, if a widget resource field is declared in a superclass, that field is included in the resource list of the superclass and does not need to be included in the resource list of the subclass. For example, since the Core class contains a resource entry for *background_pixel*, the implementation of the Label widget does not need a resource entry for *background_pixel* as well. However, a subclass can override the resource entry for any field declared in a superclass by specifying a resource entry for that field in its own resource list. To provide new values that supersede the superclass defaults, override the resource entry.

A widget does not do anything to get its own resources; **XtCreateWidget** automatically gets resources for widgets before calling the class initialize procedure.

Some widgets have subparts that are not widgets but for which the widget needs to get resources. For example, the *Text* widget gets resources for its *source* and *sink*. Widgets with such needs call **XtGetSubresources**.

Some widgets also need resources that are not specific to a widget but that apply to the overall application. In this case, widgets call **XtGetApplicationResources**.

¦ Copyright IBM Corp. 1985, 1991

4.14.3 Converting Resources

The Intrinsics provide a mechanism for registering representation converters that are automatically invoked by the resource calling routines. In addition, the Intrinsics provide and register several commonly used converters. The resource conversion mechanism serves several purposes:

It permits user and application resource files to contain ASCI representations of non-textual values.
It allows textual or other representations of default resource value that are dependent upon the display, screen, or colormap, and thus must be computed at run time.
It caches all conversion source and result data. Conversions tha require much computation or space (for example, string to translation table) or that require round trips to the server (for example, string to font or color) are performed only once.

¦ Copyright IBM Corp. 1985, 1991

4.15 Predefined Resource Converters

The Intrinsics defines all the representations used in the Core, Composite, Constraint, and Shell widgets. Furthermore, it registers resource converters for the following:

From **XtRString** to:

XtRAcceleratorTable, XtRBoolean, XtRBool, XtRCursor, XtRDimension, XtRDisplay, XtRFile, XtRFloat, XtRFont, XtRFontStruct, XtRInt, XtRPixel, XtRPosition, XtRShort, XtRTranslationTable, and **XtRUnsignedChar.**

From **XtRColor** to:

XtRPixel.

From **XtRInt** to:

XtRBoolean, XtRBool, XtRColor, XtRDimension, XtRFloat, XtRFont, XtRPixel, XtRPixmap, XtRPosition, XtRShort, and **XtRUnsignedChar.**

From **XtRPixel** to:

XtRColor.

The string to pixel conversion has two predefined constants that work in contrast with each other (**XtDefaultForeground** and **XtDefaultBackground**). These constants evaluate the black and white pixel values of the widget's screen, respectively. If, however, the application uses reverse video, they evaluate the white and black pixel values of the widget's screen, respectively.

The string to font and string to font structure converters recognize the constant **XtDefaultFont** and evaluate this to the font in the default graphics context of the screen.

Subtopics
4.15.1 Writing a New Resource Converter
4.15.2 Registering a New Resource Converter
4.15.3 Invoking Resource Converters
4.15.4 Formatting Resource Files

¦ Copyright IBM Corp. 1985, 1991

4.15.1 Writing a New Resource Converter

Type converters use pointers to **XrmValue** structures (defined in **<X11/Xresource.h>**) for input and output values.

```
typedef struct {
      unsigned int size;
      caddr_t addr;
} XrmValue, *XrmValuePtr;
```

A resource converter procedure is of type **XtConverter**. For the definition of this data type, see "XtConverter" in topic 4.17.63.

Type converters should perform the following actions:

- Check that the number of arguments passed is correct
- Attempt the type conversion
- Return a pointer to the data in the *to* parameter if successful. Otherwise, call **XtWarningMsg** and return without modifying the *to* parameter.

Most type converters take the data described by the specified *from* argument and return data by writing into the specified *to* argument. Some converters need other information, available in the specified *args*.

A type converter can invoke another type converter so that differing sources that may convert into a common intermediate result can make maximum use of the type converter cache.

The address written in **to->addr** should not be a local variable of the converter because this is not valid when the converter returns. The address should be a pointer to a static variable, as shown in the following example where *screenColor* is returned.

This example is of a converter that takes a string and converts it to a pixel.

```
  static void CvtStringToPixel(args, num_args, fromVal, toVal)
     XrmValue *args;
     Cardinal *num_args;
     XrmValue *fromVal;
     XrmValue *toVal;
  {
     static XColor screenColor;
     XColor exactColor;
     Screen *screen;
     Colormap colormap;
     Status status;
     char message[1000];
     XrmQuark q;
     String params[1];
     Cardinal num_params = 1;

     if (*num_args != 2)
     XtErrorMsg("cvtStringToPixel","wrongParameters","XtToolkitError",
         "String to pixel conversion needs screen and colormap arguments",
         (String *)NULL,(Cardinal *)NULL);
```

¦ Copyright IBM Corp. 1985, 1991

```
    screen = *((Screen **) args[0].addr);
    colormap = *((Colormap *) args[1].addr);

    LowerCase((char *) fromVal->addr,message);
    q = XrmStringToQuark(message);

    if(q == XtQExtdefaultbackground) { done(&screen->white_pixel,
      Pixel);return; }
    if(q == XtQExtdefaultforeground) { done(&screen->black_pixel,
      Pixel);return; }
    if ((char) fromVal->addr&lrbk.0] == '#') { /* a color definition */

        status = XParseColor(DisplayOfScreen(screen),colormap,
            (String)fromVal->addr, &screenColor);
        if(status != 0) status = XAllocColor(DisplayOfScreen(screen),
            colormap,&screenColor);

    } else /* some color name */

    status = XAllocNamedColor(DisplayOfScreen(screen),colormap,
             (String)fromVal->addr, &screenColor,&exactColor);

    if(status == 0) {

        params[0]=(String)fromVal->addr;
        XtWarningMsg("cvtStringToPixel","noColormap",
                     "XtToolkitError",
            "Cannot allocate colormap entry for\"%s\"",
                 params,&num_params);

    } else {

        toVal->addr = (caddr_t)&screenColor.pixel;
        toVal->size = sizeof(Pixel);

  }
  };
```

All type converters should define some set of conversion values that they will succeed on so these can be used in the resource defaults. This issue arises only with conversions where there is no string representation that all server implementations will necessarily recognize, such as fonts and colors. For resources of this nature, the converter should define a symbolic constant, such as **XtDefaultForeground**, **XtDefaultBackground**, or **XtDefaultFont**.

¦ Copyright IBM Corp. 1985, 1991

4.15.2 Registering a New Resource Converter

To register a new resource converter, use the **XtAppAddConverter** routine. See "XtAppAddConverter" in topic 4.17.19 for a description of this routine.

For the few type converters that need additional arguments, the Intrinsics conversion mechanism provides a method of specifying how these arguments should be computed. The enumerated type **XtAddressMode** and the structure **XtConvertArgRec** specify how each argument is derived. Both the structure and the type are defined in the **<X11/Convert.h>** header file.

```
typedef enum {


/* address mode              parameter representation        */
   XtAddress,                /* address                        */
   XtBaseOffset,             /* offset                         */
   XtImmediate,              /* constant                       */
   XtResourceString,         /* resource name string           */
   XtResourceQuark           /* resource name quark            */

}   XtAddressMode;


typedef struct {
    XtAddressMode address_mode;
    caddr_t address_id;
    Cardinal size;
} XtConvertArgRec, *XtConvertArgList;
```

The *address_mode* specifies how the *address_id* should be interpreted. The *size* specifies the length of the data in bytes.

XtAddress Causes *address_id* to be interpreted as the address of the data.

XtBaseOffset Causes *address_id* to be interpreted as the offset from the widget base.

XtImmediate Causes *address_id* to be interpreted as a constant.

XtResourceString Causes *address_id* to be interpreted as the name of a resource that is to be converted into an offset from widget base.

XtResourceQuark Is an internally compiled form of an **XtResourceString**.

The code for registering the preceding **CvtStringToPixel** routine is:

```
static XtConvertArgRec colorConvertArgs[.] = {
    {XtBaseOffset, (caddr_t) XtOffset(Widget, core.screen),
                   sizeof(Screen*)}
    {XtBaseOffset, (caddr_t) XtOffset(Widget, core.colormap),
                   sizeof(Colormap)}
};

XtAddConverter(XtRString, XtRPixel, CvtStringToPixel,
                   colorConvertArgs, XtNumber(colorConvertArgs));
```

¦ Copyright IBM Corp. 1985, 1991

The conversion argument descriptors *colorConvertArgs* and *screenConvertArg* are predefined.

The *screenConvertArg* descriptor puts the screen field for the widget into *args[0]* field.

The *colorConvertArgs* descriptor puts the screen field of the widget into *args[0]* field, and the widget *colormap* field into *args[1]* field.

Even though it appears easier to create a descriptor that puts the base address for the widget into *args[0]* field and perform the indexing in the conversion routine, it is not encouraged. Constraining the dependencies of the conversion procedure to a minimum increases the chance that subsequent conversions will find what they need in the conversion cache. Instead, decrease the size of the cache by having fewer, but more widely applicable entries.

¦ Copyright IBM Corp. 1985, 1991

4.15.3 Invoking Resource Converters

Resource-calling routines call resource converters if the user specifies a resource that is a different representation from the desired representation or if the widget's default resource value representation is different from the desired representation. Examples of resource-calling routines are **XtGetSubresources** and **XtGetApplicationResources**.

To invoke conversions, use **XtConvert** or **XtDirectConvert**.

Subtopics
4.15.3.1 Reading and Writing Widget State

¦ Copyright IBM Corp. 1985, 1991

4.15.3.1 Reading and Writing Widget State

Any resource field in a widget can be read or written by a client. On a write, the widget decides what changes it allows and updates all derived fields appropriately.

The following list describes the routines to use for various widget state operations.

XtGetValues — Retrieves the current value of a resource associated with a widget instance. See "XtGetValues" in topic 4.17.98.

XtGetSubvalues — Retrieves the current value of nonwidget resource data associated with a widget instance. See "XtGetSubvalues" in topic 4.17.97.

XtSetValues — Modifies the current value of a resource associated with a widget instance. See "XtSetValues" in topic 4.17.169.

XtSetSubvalues — Sets the current value of a nonwidget resource associated with a widget instance. See "XtSetSubvalues" in topic 4.17.168.

¦ Copyright IBM Corp. 1985, 1991

4.15.4 Formatting Resource Files

A resource file contains text representing the default resource values for an application or set of applications. The resource file is an ASCII text file that consists of a number of lines with the following EBNF syntax:

```
Xdefault        = {line "\n"}.

line            = (comment | production).

comment         = "!" string.

production      = resourcename ":" string.

resourcename    = ["*"] name {("." | "*") name}.

string          = {<any character not including eol>}.

name            = {"A"-"Z" | "a"-"z" | "0"-"9"}.
```

If the last character on a line is a backslash (\), that line is assumed to continue on the next line. To include a new-line character in a string, use "-n".

¦ Copyright IBM Corp. 1985, 1991

4.16 Using Translation Management

Except under unusual circumstances, widgets do not hardwire the mapping of user events into widget behavior by using the **Event Manager**. Instead, they provide a default that can be overridden by the user.

The **Translation Manager** provides an interface to specify and manage the mapping of X-Windows event sequences into widget-supplied functionality. For example, call procedure **Abc** when the **Y** key is pressed.

The **Translation Manager** uses two kinds of tables to perform translations.

> The ***action table***, which is in the widget class structure, specifies the mapping of externally available procedure name strings to the corresponding procedure implemented by the widget class.
>
> The ***translation table***, which is also in the widget class structure, specifies the mapping of event sequence to procedure name strings.

The translation table in the class structure can be overridden for a specific widget instance by supplying a different translation table for the widget instance. The resource name is **XtNtranslations**.

Subtopics

¦ Copyright IBM Corp. 1985, 1991

4.16.1 Using Action Tables

All widget class records contain an action table. In addition, an application can register its own action tables with the **Translation Manager**, so that the translation tables it provides to widget instances can access application functionality. The action table of the application uses the following structure:

```
typedef struct _XtActionsRec {
      String action_name;
      XtActionProc action_proc;
} XtActionsRec, *XtActionList;
```

The *action_name* is used in translation tables to access the procedure. The translation procedure *action_proc* is a pointer of type **XtActionProc**, which points to a procedure that implements the functionality.

For example, the Command widget has procedures to:

```
    Set the command button to indicate it is activate
    Reset the command button to its normal mod
    Highlight the button border
    Unhighlight the button border
    Notify any callbacks that the button has been activated
```

The action table for the Command widget class makes these functions available to translation tables written for Command or any subclass. The string entry is the name used in translation tables.

The procedure entry, which is usually spelled the same as the string, is the name of the C language procedure that implements that function:

```
XtActionsRec actionTable[] = {
     {"Set",  Set},
     {"Unset",  Unset},
     {"Highlight",  Highlight},
     {"Unhighlight",  Unhighlight},
     {"Notify",  Notify}
};
```

To declare an action table and register it with the **Translation Manager**, use the **XtAppAddActions** routine.

¦ Copyright IBM Corp. 1985, 1991

4.16.2 Translating Action Names to Procedures

The **Translation Manager** uses a simple algorithm to convert the name of the procedure specified in a translation table to the procedure specified in an action table. When the widget is realized, it performs a search for the name in the following tables:

The widget class action table for the nam

The widget superclass action table and others up the superclass chai

The action tables registered with **XtAddActions**, from the latest table added to the earliest.

The **Translation Manager** stops the search as soon as it finds a name. If no name is found, it generates an error.

¦ Copyright IBM Corp. 1985, 1991

4.16.3 Using Translation Tables

All widget instance records contain a translation table. A translation table specifies the action procedures invoked for an event or a sequence of events. The table is a string containing a list of translations from an event or an event sequence to procedure calls. The translations are separated from one another by new-line characters (ASCII LF). The translation table has no default value.

For example, the default behavior of Command is:

```
Highlight on enter windo
Unhighlight on exit windo
Invert on left button dow
Call callbacks and reinvert on left button up
```

The default translation table for Command is:

```
  static String defaultTranslations =

        "<EnterWindow>:    Highlight()\n\
         <LeaveWindow>:    Unhighlight()\n\
         <Btn1Down>:       Set()\n\
         <Btn1Up>:         Notify() Unset()";
```

For details on the syntax of translation tables, see "Translation Table File Syntax" in topic 4.16.8.

The *tm_table* of the **CoreClass** record should be filled in at the time of static initialization with the string containing the default translations of the class. If a class must inherit the translations of its superclass, it can store the special value **XtInheritTranslations** into *tm_table*. After the class initialization procedures have been called, the Intrinsics compile this translation table into an efficient internal form. Then, at widget creation time, this default table is used for any widgets that have not had their Core translations field set by the **Resource Manager** or the initialize procedures.

The resource conversion mechanism takes care of automatically compiling string translation tables that are resources. If a client uses translation tables that are not resources, it must compile them using **XtParseTranslations**.

The Intrinsics use the compiled form of the translation table to register the necessary events with the **Event Manager**. Widgets need to specify only the action and translation tables for events to be processed by the **Translation Manager**.

Subtopics
4.16.3.1 Event Sequences
4.16.3.2 Action Sequences

¦ Copyright IBM Corp. 1985, 1991

4.16.3.1 Event Sequences

An event sequence is a comma-separated list of X-Windows event descriptions that describes a specific order of X events to map to a set of program actions. Each of these event descriptions consists of three parts:

The X event typ
A prefix consisting of the modifier bit
An event-specific suffi

Various abbreviations can be used to make translation tables easier to read.

¦ Copyright IBM Corp. 1985, 1991

4.16.3.2 Action Sequences

Action sequences specify what program or widget actions to take in response to incoming events. An action sequence contains a series of action procedure call specifications, each of which gives the name of an action procedure and a list of string parameters, in parentheses, to pass to that procedure.

¦ Copyright IBM Corp. 1985, 1991

4.16.4 Merging Translation Tables

Sometimes an application needs to add its own translations in place of or in addition to the translation of the widget. For example, a window manager provides functions to move a window. Normally, this window manager can move the window when any pointer button is pressed down in a title bar. It allows the user to specify other translations for the middle or right button down in the title bar, but it ignores any user translations for left button down.

To accomplish this, the window manager should first create the title bar and then merge the two translation tables into the title bar translations. One translation table contains the translations that the window manager wants if the user has not specified a translation for a particular event or event sequence. The other translation table contains the translations that the window manager wants regardless of what the user has specified.

Three functions support this merging:

XtParseTranslationTable compiles a translation table.

XtAugmentTranslations nondestructively merges a compiled translation table into the compiled translation table of a widget.

XtOverrideTranslations destructively merges a compiled translation table into the compiled translation table of a widget.

¦ Copyright IBM Corp. 1985, 1991

4.16.5 Changing Translation Tables

To replace a widget's translations completely, use the **XtSetValues** routine on the **XtNtranslations** resource and specify a compiled translation table as the value.

To make it possible to easily modify translation tables in resource files, the string-to-translation-table resource type converter allows specification of whether the table replaces, augments, or overrides any existing translation table in the widget. To do this, the first character in the table should be a number sign (#), followed by **replace** (the default action), **augment**, or **override**, according to the application's needs.

To completely remove existing translations, use the **XtUninstallTranslations** routine.

¦ Copyright IBM Corp. 1985, 1991

4.16.6 Using Accelerators

Applications often need to be able to bind events in one widget to actions in another, such as invoking menu actions from the keyboard. To accomplish this, the Intrinsics provide ***accelerators***. An accelerator is a translation table that is bound with its actions in the context of a particular widget. The accelerator table can then be installed on a destination widget, which allows an action in the destination widget to execute an accelerator action as if it were triggered in the accelerator widget.

Each widget instance contains that widget's exported accelerator table. Each class of widget exports a method that takes a displayable string representation of the accelerators so that the widgets can display their current accelerators. The representation is the accelerator table in canonical representation form, as described in "Canonical Representation" in topic 4.16.8.6.

Use these routines to perform the following accelerator function:

XtParseAcceleratorTable	Parses an accelerator table.
XtInstallAccelerators	Installs accelerators from a widget on another widget.
XtInstallAllAccelerators	Installs all accelerators from a widget and all of its descendants onto one destination.

¦ Copyright IBM Corp. 1985, 1991

4.16.7 Converting Key Codes to KeySyms

The translation manager provides support for automatically translating key codes in incoming key events to KeySyms. Pointers for key code to KeySym translation procedures are of type **XtKeyProc**. The following key code translation operations are possible:

XtSetKeyTranslator	Registers a key translator.
XtRegisterCaseConverter	Registers a case converter. Pointers to case conversion procedures are of type **XtCaseProc**.
XtConvertCase	Determines the upper- and lowercase equivalents for a KeySym.

¦ Copyright IBM Corp. 1985, 1991

4.16.8 Translation Table File Syntax

A translation table file is an ASCII text file. This section explains the proper notation and syntax as well as common modifier names, event type values, supported abbreviations, canonical representations, and examples.

Subtopics

¦ Copyright IBM Corp. 1985, 1991

4.16.8.1 Notation

Syntax is specified in EBNF notation, where:

```
     [ a ]
```

means either "a" or nothing and

```
     { a }
```

means "a" has zero or more occurrences.

All terminals are enclosed in double quotes (" "). Informal descriptions are enclosed in <angle> brackets.

¦ Copyright IBM Corp. 1985, 1991

4.16.8.2 Syntax

The syntax of the translation table file is:

translationTable = [directive] { production }

directive = ("#replace" | "#override" | "#augment") "\"

production = lhs ":" rhs "\n"

lhs = (event|keyseq) { "," (event|keyseq) }

keyseq = """keychar {keychar} """

keychar = ["^"|"$"] <ascii character>

event = [modifier_list] "<"event_type">" ["("count["+"]")"] {detail}

modifier_list = (["!"|":"] {modifier})|"None"

modifier = ["~"] modifier_name

count = ("1" | "2" | "3" | "4" | ...)

modifier_name = "@" <keysym> | <see ModifierNames table below>

event_type = <see Event Types table below>

detail = <event specific details>

rhs = {name "(" [params] ")" }

name = namechar { namechar }

namechar = { "a"-"z" | "A"-"Z" | "0"-"9" | "$" | "_" }

params = string {"," string}

string = quoted_string | unquoted_string

quoted_string = """ {<ASCII character>} """

unquoted_string = {<ASCII character except space, tab, ",", newline, ")">}

It is often convenient to include newlines in a translation table to make it more readable. In C language, the newline should be preceded by a backslash (\):

```
"<Btn1Down>:      DoSomething()\n\
 <Btn2Down>:        DoSomethingElse()"
```

¦ Copyright IBM Corp. 1985, 1991

4.16.8.3 Modifier Names

The *modifier* field is used to specify normal X-Windows keyboard and button modifier mask bits. Modifiers are allowed on the following event types and their abbreviations:

KeyPress MotionNotify
KeyRelease EnterNotify
ButtonPress LeaveNotify
ButtonRelease

An error occurs when a translation table that contains modifiers for any other events is parsed.

Modifiers have the "don't care" value if the *modifier_list* has no entries and is not **None**.

Listed modifiers must be in the correct state and "don't care" abou other modifiers if any modifiers are specified and **!** is not specified.

Listed modifiers must be in the correct state and no other modifier can be asserted if **!** is specified at the beginning of the modifier list.

Modifiers preceded by a "~" (tilde) must not be asserted

Modifiers cannot be asserted if **None** is specified.

Standard modifiers in the event, which map the event key code to keysym, are applied by the Intrinsics when a colon (:) is specified at the beginning of the modifier list.

The default standard modifiers are **Shift** and **Lock**. The resulting keysym must exactly match the specified keysym, and the nonstandard modifiers in the event must match the **modifier_list**. For example, ":<Key>a" is different from "<Key>A" and ":Shift<Key>A" is different from ":<Key>A".

No standard modifiers are applied when a colon (:) is not specified This makes "<Key>A" and "<Key>a", for example, equivalent.

In key sequences, a circumflex (^) is an abbreviation for the Control modifier; a dollar sign ($) is an abbreviation for Meta; and a backslash (\) is used to quote any character, such as a double quote ("), a circumflex (^), a dollar sign ($), or another backslash (\).

Using **None** for a **modifier_list** is the same as using an exclamation point with no modifiers.

In brief:

No Modifiers:	**None** <event> detail
Any Modifiers:	<event> detail
Only these Modifiers:	**!** mod1 mod2 <event> detail
These modifiers and any others:	mod1 mod2 <event> detail

Modifier	**Abbreviation**	**Meaning**

¦ Copyright IBM Corp. 1985, 1991

Ctrl	**c**	**Control** modifier bit
Shift	**s**	**Shift** modifier bit
Lock	**l**	**Lock** modifier bit
Meta	**m**	**Meta** key modifier*
Hyper	**h**	**Hyper** key modifier*
Super	**su**	**Super** key modifier*
Alt	**a**	**Alt** key modifier*
Mod1		**Mod1** modifier bit
Mod2		**Mod2** modifier bit
Mod3		**Mod3** modifier bit
Mod4		**Mod4** modifier bit
Mod5		**Mod5** modifier bit
Button1		**Button1** modifier bit
Button2		**Button2** modifier bit
Button3		**Button3** modifier bit
Button4		**Button4** modifier bit
Button5		**Button5** modifier bit
ANY		Any combination

*A key modifier is any modifier bit whose corresponding key code contains the corresponding left or right keysym.

For example, **m** or **Meta** means any modifier bit that maps to a key code whose keysym list contains **XK_Meta_L** or **XK_Meta_R**. This interpretation applies for each display, not globally or for each application context. The **Control**, **Shift**, and **Lock** modifier names refer explicitly to the corresponding modifier bits, and there is no additional interpretation of keysyms for these modifiers.

Because arbitrary keysyms can be associated with modifiers, the set of modifier key modifiers is extensible. The "@" <keysym> syntax means any modifier bit whose corresponding key code contains the specified keysym.

A modifier_list/keysym combination in a translation is matched with a modifiers/keycode combination in an event as follows:

1. If a colon (:) is used, the Intrinsics call the **XtKeyProc** for the display with the keycode and modifiers. For a match to occur, **(***modifiers* **&** ***~modifiers_return*****)** must equal *modifier_list* and *keysym_return* must equal the given keysym.

2. If a colon is not used, the Intrinsics mask off all "don't care" bits

¦ Copyright IBM Corp. 1985, 1991

from the modifiers. The value must be equal to *modifier_list*. Then, for each possible combination of "don't care" modifiers in the *modifier_list*, the Intrinsics call the display's **XtKeyProc** with the keycode and that combination OR'd with the cared-about modifier bits from the event. The *keysym_return* must match the keysym in the translation.

¦ Copyright IBM Corp. 1985, 1991

4.16.8.4 Event Types

The *EventType*, which are defined below, describes the **XEvent** types.

Event Type	**Value**
KeyKeyDown	KeyPress
KeyUp	KeyRelease
BtnDown	ButtonPress
BtnUp	ButtonRelease
Motion **PtrMoved** **MouseMoved**	MotionNotify
EnterEnterWindow	EnterNotify
Leave **LeaveWindow**	LeaveNotify
FocusIn	FocusIn
FocusOut	FocusOut
Keymap	KeymapNotify
Expose	Expose
GrExp	GraphicsExpose
NoExp	NoExpose
Visible	VisibilityNotify
Create	CreateNotify
Destroy	DestroyNotify
Unmap	UnmapNotify
Map	MapNotify
MapReq	MapRequest
Reparent	ReparentNotify
Configure	ConfigureNotify

Copyright IBM Corp. 1985, 1991

ConfReq	ConfigureRequest
Grav	GravityNotify
ResReq	ResizeRequest
Circ	CirculateNotify
CircReq	CirculateRequest
Prop	PropertyNotify
SelClr	SelectionClear
SelReq	SelectionRequest
Select	SelectionNotify
Clrmap	ColormapNotify
Message	ClientMessage
Mapping	MappingNotify

Copyright IBM Corp. 1985, 1991

4.16.8.5 Supported Event Type Abbreviations

The *detail* is event-specific and corresponds normally to the *detail* of an **XEvent**, for example, <key>**A**. If no *detail* is specified, then **ANY** is assumed.

Abbreviation	Meaning
Ctrl	**KeyPress** with control modifier
Meta	**KeyPress** with meta modifier
Shift	**KeyPress** with shift modifier
Btn1Down	**ButtonPress** with **Btn1** detail
Btn1Up	**ButtonRelease** with **Btn1** detail
Btn2Down	**ButtonPress** with **Btn2** detail
Btn2Up	**ButtonRelease** with **Btn2** detail
Btn3Down	**ButtonPress** with **Btn3** detail
Btn3Up	**ButtonRelease** with **Btn3** detail
Btn4Down	**ButtonPress** with **Btn4** detail
Btn4Up	**ButtonRelease** with **Btn4** detail
Btn5Down	**ButtonPress** with **Btn5** detail
Btn5Up	**ButtonRelease** with **Btn5** detail
BtnMotion	**MotionNotify** with any button modifier
Btn1Motion	**MotionNotify** with **Button1** modifier
Btn2Motion	**MotionNotify** with **Button2** modifier
Btn3Motion	**MotionNotify** with **Button3** modifier
Btn4Motion	**MotionNotify** with **Button4** modifier
Btn5Motion	**MotionNotify** with **Button5** modifier

A keysym can be specified as any of the standard keysym names; as a hexadecimal number prefixed with **0x** or **0X**; as an octal number prefixed with **0**; or as a decimal number. A keysym expressed as a single digit is interpreted as the corresponding ASCII keysym. For example, **0** is the keysym **XK_0**. Other single character keysyms are treated as literal constants from ASCII. For example, **!** is treated as **0x21**. Standard keysym names are those defined in **<X11/keysymdef.h>**, with the **XK_** prefix removed.

¦ Copyright IBM Corp. 1985, 1991

4.16.8.6 Canonical Representation

Every translation table has a unique, canonical text representation. This representation is passed to a widget's **display_accelerator** method to describe the accelerators installed on that widget. The syntax of a translation table file is described in "Syntax" in topic 4.16.8.2. The canonical representation of a translation table is:

translationTable = { production }

production = lhs ":" rhs "\n"

lhs = event { "," event }

event = [modifier_list] "<"event_type">" ["("count["+"]")"] {detail}

modifier_list = (["!"|":"] {modifier})

modifier = ["~"] modifier_name

count = ("1" | "2" | "3" | "4" | ...)

modifier_name = "@" <keysym> | <see canonical modifier names below>

event_type = <see canonical event types below>

detail = <event specific details>

rhs = {name "(" [params] ")" }

name = namechar { namechar }

namechar = { "a"-"z" | "A"-"Z" | "0"-"9" | "$" | "_" }

params = string {"," string}

string = quoted_string

quoted_string = """ {<ASCII character>} """

The canonical modifier names are:

Ctrl	Button1	Mod1
Lock	Button2	Mod2
Shift	Button3	Mod3
	Button4	Mod4
	Button5	Mod5

The canonical event types are:

KeyPress	**KeyRelease**	**ButtonPress**
ButtonRelease	**MotionNotify**	**LeaveNotify**
EnterNotify	**KeymapNotify**	**FocusIn**
FocusOut	**Expose**	**GraphicsExpose**
NoExpose	**VisibilityNotify**	**CreateNotify**
DestroyNotify	**UnmapNotify**	**MapNotify**
MapRequest	**ReparentNotify**	**ConfigureNotify**
ConfigureRequest	**GravityNotify**	**ResizeRequest**

¦ Copyright IBM Corp. 1985, 1991

CirculateNotify	**CirculateRequest**	**PropertyNotify**
SelectionClear	**SelectionRequest**	**SelectionNotify**
ColormapNotify	**ClientMessage**	

¦ Copyright IBM Corp. 1985, 1991

4.16.8.7 Examples of Event Types

Put more specific events in the table before more general events

Shift <Btn1Down> : *twas()\n\.*
<Btn1Down> : *brillig()*

For double-click on **Button 1 Up** with **Shift**, use:

Shift<Btn1Up>(2) : *and()*

This is equivalent to

Shift<Btn1Down>,Shift<Btn1Up>,Shift<Btn1Down>,Shift<Btn1Up> : *and()*

with appropriate timers set between events.

For double-click on **Button 1 Down** with *Shift* use:

Shift<Btn1Down>(2) : *the()*

It is equivalent to:

Shift<Btn1Down>,Shift<Btn1Up>,Shift<Btn1Down> : *the()*

with appropriate timers set between events.

Mouse motion is always discarded when it occurs between events in table where no motion event is specified:

<Btn1Down>,<Btn1Up> : *slithy()*

This is taken, even if the mouse jiggles a bit between the down and up events. Similarly, any motion event specified in a translation matches any number of motion events. If the motion event causes an action procedure to be invoked, the procedure is invoked after each motion event.

If an event sequence consists of a sequence of events that is also non-initial subsequence of another translation, it is not taken if it occurs in the context of the longer sequence. This usually occurs in the following types of sequences:

<Btn1Down>,<Btn1Up> : *toves()\n*
<Btn1Up> : *did()*

The second translation is taken only if the button release is not preceded by a button press or if there are intervening events between the press and release sequence. This precaution is especially useful when using the repeat notation with buttons and keys and when specifying motion events because they are included between any two other events.

Note: Mouse motion and double-click translations cannot exist in the same translation table.

For single click on **Button 1 Up** with **Shift** and **Meta**, use:

Shift Meta<Btn1Down>,Shift Meta<Btn1Up>:*gyre()*

¦ Copyright IBM Corp. 1985, 1991

The "+" notation allows you to say "for any number of clicks greate than or equal to count", such as:

Shift <Btn1Up>(2+) : *and()*

To say **EnterNotify** with any modifiers, use:

<Enter> : *gimble()*

To say **EnterNotify** with no modifiers, use:

None <Enter> : *in()*

To say **EnterNotify** with **Button 1 Down** and **Button 2 Up** and don't care about the other modifiers, use:

Button1 ~Button2 <Enter> : *the()*

To say **EnterNotify** with **Button1 Down** and **Button2 Down** exclusively, use:

! Button1 Button2 <Enter> : *wabe()*

Using "~" with "!" is not necessary.

¦ Copyright IBM Corp. 1985, 1991

4.17 Toolkit Routines and Procedures

The Toolkit routines and procedures are described in the remainder of this chapter. These routines and procedures are in alphabetical order.

Subtopics

¦ Copyright IBM Corp. 1985, 1991

¦ Copyright IBM Corp. 1985, 1991

¦ Copyright IBM Corp. 1985, 1991

¦ Copyright IBM Corp. 1985, 1991

4.17.1 MenuPopdown

void MenuPopdown(*shell_name*)
     String *shell_name*;

shell_name Specifies the name of the widget shell to popdown.

MenuPopdown pops down a spring-loaded menu when a pointer button is released or when the pointer is moved into some window.

If a shell name is not specified, **MenuPopdown** calls **XtPopdown** with the widget for the specified translation.

If a shell name is specified in the translation table, **MenuPopdown** tries to find the shell by searching the widget tree. It starts at the parent of the widget where it was invoked. If it finds a shell with the specified name in the pop-up children of that parent, it pops down the shell. Otherwise, it moves up the parent chain as needed.

If **Menudown** gets to the application top-level widget and cannot find a matching shell, it generates an error.

¦ Copyright IBM Corp. 1985, 1991

4.17.2 MenuPopup

void MenuPopup(*shell_name*)
String *shell_name*;

shell_name Specifies the name of the widget shell to pop up.

MenuPopup pops up a menu when a pointer button is pressed or when the pointer is moved into a window.

MenuPopup is known to the translation manager, which must perform special actions for spring-loaded pop-ups. Calls to **MenuPopup** in a translation specification are mapped into calls to a non-exported action procedure. The translation manager fills in parameters based on the event specified on the event specified on the lefthand side of a translation.

If **MenuPopup** is invoked upon **ButtonPress**, possibly with modifiers, the translation manager pops up the shell with *grab_kind* set to **XtGrabExclusive** and *spring_loaded* set to **True**.

If **MenuPopup** is invoked upon **EnterWindow**, possibly with modifiers, the translation manager pops up the shell with *grab_kind* set to **XtGrabNonexclusive** and *spring_loaded* set to **False**. Otherwise, the translation manager generates an error.

When the widget is popped up, **MenuPopup** does the following:

Calls **XtCheckSubclass** to ensure that the *popup_shell* is a subclass of **Shell**

Generates an error if the *popped_up* field of the shell is already **True**

Calls the callback procedures on the *popup_callback* list of the shell

Sets the shell *popped_up* field to **True**

Sets the shell *grab_kind* and *spring_loaded* fields appropriately

Calls the *create_popup_child* of the shell with *popup_shell* as the parameter if the *create_popup_child* of the shell is non-null

Calls the following

XtAddGrab(*popup_shell, (grab_kind* == **XtGrabExclusive**), *spring_loaded*)

Calls **XtRealizeWidget** with the specified popup_shell

Calls **XMapWindow** with the specified popup_shell.

These actions are the same for **XtPopup**.

MenuPopup tries to find the shell by looking up the widget tree starting at the parent of the widget in which this routine is invoked. If it finds a shell with the specified name in the pop-up children of that parent, it pops up the shell with the appropriate parameters. Otherwise, it moves up the parent chain as needed. **MenuPopup** generates an error if it gets to the application widget and cannot find a matching shell.

¦ Copyright IBM Corp. 1985, 1991

4.17.3 XtAcceptFocusProc

```
typedef Boolean(*XtAcceptFocusProc)(Widget, Time);
      Widget widget;
      Time *time;
```

widget Specifies the widget.

time Specifies the X time of the event causing the accept focus.

XtAcceptFocusProc is the pointer for an *accept_focus* procedure. To allow outside agents to cause a widget to get the input focus, every widget exports an *accept_focus* procedure. The widget returns whether or not it accepts the focus, so that the parent can give the focus to another widget.

Widgets that must know when they lose the input focus should use the **Xlib** focus notification mechanism explicitly by specifying translations for **FocusIn** and **FocusOut** events.

Widgets that do want the input focus should set their *accept_focus* procedure pointer to **NULL**.

Widgets that need the input focus can call **XSetInputFocus** explicitly.

¦ Copyright IBM Corp. 1985, 1991

4.17.4 XtActionProc

```
typedef void (*XtActionProc)(Widget, XEvent*, String*, Cardinal*);
      Widget widget;
      XEvent *event;
      String *params;
      Cardinal *num_params;
```

widget Specifies the widget that caused the action.

event Specifies the event that caused the action. If the action is called after a sequence of events, then the last event in the sequence is used.

params Specifies a pointer to the list of strings specified in the translation table as arguments to the action.

num_params Specifies the number of arguments specified in the translation table.

XtActionProc is the pointer for the the translation *action_proc* procedure.

All widget class records contain an action table. In addition, an application can register its own action tables with the translation manager so that the translation tables it provides to the widget instances can access application functionality by using this procedure.

¦ Copyright IBM Corp. 1985, 1991

4.17.5 XtActionsRec

```
typedef struct_XtActionsRec{
     String action_name;
     XtActionProc action_proc;
}XtActionsRec, *XtActionList;
```

action_name Specifies the procedure in the translation tables.

action_proc Specifies a pointer to a procedure that implements the application functionality.

¦ Copyright IBM Corp. 1985, 1991

4.17.6 XtAddActions

void XtAddActions(*actions, num_actions*)
XtActionList *actions*;
Cardinal *num_actions*;

actions Specifies the action table to be registered.

num_args Specifies the number of entries in this action table.

XtAddActions declares an action table and registers it with the translation manager. If more than one action is registered with the same name, the most recently registered action is used. If duplicate actions exist in an action table, the first entry is used.

The Intrinsics registers an action table for **MenuPopup** and **MenuPopdown** as part of the Toolkit initialization.

¦ Copyright IBM Corp. 1985, 1991

4.17.7 XtAddCallback

void XtAddCallback(*widget, callback_name, callback, client_data*)
Widget *widget*;
String *callback_name*;
XtCallbackProc *callback*;
caddr_t *client_data*;

widget Specifies the widget.

callback_name Specifies the callback list where the procedure will be appended.

callback Specifies the callback procedure.

client_data Specifies the argument for callback if the callback is invoked by **XtCallCallbacks**. Otherwise, it specifies **NULL**.

XtAddCallback adds a callback procedure to the callback list of the specified widget. A callback is invoked as many times as it occurs in the callback list.

¦ Copyright IBM Corp. 1985, 1991

4.17.8 XtAddCallbacks

void XtAddCallbacks(*widget, callback_name, callbacks*)
Widget *widget*;
String *callback_name*;
XtCallbackList *callbacks*;

widget Specifies the widget.

callback_name Specifies the callback list where the procedure will be appended.

callbacks Specifies the null-terminated list of callback procedures and corresponding client data.

XtAddCallbacks adds a list of callback procedures to the callback list of specified widget.

¦ Copyright IBM Corp. 1985, 1991

4.17.9 XtAddConverter

void XtAddConverter(*from_type, to_type, converter, convert_args, num_args*)
String *from_type*;
String *to_type*;
XtConverter *converter*;
XtConvertArgList *convert_args*;
Cardinal *num_args*;

from_type Specifies the source type.

to_type Specifies the destination type.

converter Specifies the type converter procedure.

convert_args Specifies how to compute the additional arguments to the converter.

num_args Specifies the number of additional arguments to the converter.

XtAddConverter registers a new converter. If the type converter does not require additional arguments, *convert_args* is **NULL** and *num_args* is zero.

For converters that require additional arguments, use **XtAddressMode** and **XtConvertArgRec** to specify how each argument is derived. (See <**X11/Convert.h**>.)

¦ Copyright IBM Corp. 1985, 1991

4.17.10 XtAddEventHandler

void XtAddEventHandler(*widget, event_mask, nonmaskable, proc, client_data*)
Widget *widget*;
EventMask *event_mask*;
Boolean *nonmaskable*;
XtEventHandler *proc*;
caddr_t *client_data*;

widget Specifies the widget for the event handler.

event_mask Specifies the event mask for this procedure.

nonmaskable Specifies a Boolean value that indicates if this procedure should be called on the nonmaskable events.

proc Specifies the event handler procedure.

client_data Specifies additional data for the client event handler.

XtAddEventHandler registers an event handler procedure with the dispatch mechanism when an event matching the mask occurs on the specified widget. If the widget is realized, **XtAddEventHandler** calls **XSelectInput**, if necessary.

If the procedure is registered already with the same client data, the specified mask is ORed into the existing mask.

The nonmaskable events are **GraphicsExpose**, **NoExpose**, **SelectionClear**, **SelectionRequest**, **SelectionNotify ClientMessage** and **MappingNotify**.

¦ Copyright IBM Corp. 1985, 1991

4.17.11 XtAddExposureToRegion

void XtAddExposureToRegion(*event, region*)
    XEvent **event*;
    Region *region*;

event Specifies a pointer to the **Expose** or **GraphicsExpose** event.

region Specifies the region object.

XtAddExposureToRegion merges **Expose** and **GraphicsExpose** events into a *region* as defined in **<X11/Xutil.h>**. This routine computes the union of the rectangle defined by the specified exposure event and region. Then, it stores the results in the *region* argument.

If the *event* argument is not an **Expose** or **GraphicsExpose** event, **XtAddExposureToRegion** returns without an error and does not modify the region.

¦ Copyright IBM Corp. 1985, 1991

4.17.12 XtAddGrab

void XtAddGrab(*widget, exclusive, spring_loaded*)
Widget *widget*;
Boolean *exclusive*;
Boolean *spring_loaded*;

widget Specifies the widget to add to the modal cascade.

exclusive Specifies whether user events should be dispatched to this widget exclusively or dispatched also to previous widgets in the cascade.

spring_loaded Specifies if this widget was popped up by pressing a pointer button.

XtAddGrab redirects user input to a modal widget. It appends the widget and associated parameters to the modal cascade. **XtAddGrab** checks that *exclusive* is **True** if *spring_loaded* is **True**. Otherwise, it generates an error.

If the widget exists, regardless of where remap events occur on the screen, these events are always delivered to the most recent widget in the active subset of the cascade that has *spring_loaded* set to **True**.

¦ Copyright IBM Corp. 1985, 1991

4.17.13 XtAddInput

XtInputId XtAddInput(*source, condition, proc, client_data*)
int *source*;
caddr_t *condition*;
XtInputCallbackProc *proc*;
caddr_t *client_data*;

source Specifies the source file descriptor or other device-dependent specification.

condition Specifies the mask that indicates a read, write, or exception condition or other operating system dependent condition.

proc Specifies the callback procedure.

client_data Specifies the parameter for the specified procedure.

XtAddInput registers a new source of events, usually as file input or file output, with the X Toolkit default application. (In this case, file represents any sink or source of data.) Also, **XtAddInput** specifies the conditions under which the source can generate events. When input on this source in the default application context is pending, the callback procedure is called.

Call **XtInitialize** before using **XtAddInput**.

Note: This routine has been replaced by **XtAppAddInput**.

¦ Copyright IBM Corp. 1985, 1991

4.17.14 XtAddRawEventHandler

void XtAddRawEventHandler(*widget, event_mask, nonmaskable, proc, client_data*)
Widget *widget*;
EventMask *event_mask*;
Boolean *nonmaskable*;
XtEventHandler *proc*;
caddr_t *client_data*;

widget Specifies the widget for this event handler.

event_mask Specifies the event mask for this procedure.

nonmaskable Specifies a Boolean value that indicates if this procedure should be removed on the nonmaskable events.

proc Specifies the client event handler procedure.

client_data Specifies additional data for the client event handler.

XtAddRawEventHandler registers an event handler procedure with the dispatch mechanism without causing the server to select for that event. This routine removes a previously registered raw event handler, but does not affect the mask of the widget and does not causes an **XSelectInput** for its events. Note that the widget might already have those mask bits set because of other non-raw event handlers registered on it.

The nonmaskable events are **GraphicsExpose, NoExpose, SelectionClear, SelectionRquest, SelectionNotify, ClientMessage**, and **MappingNotify**.

¦ Copyright IBM Corp. 1985, 1991

4.17.15 XtAddTimeOut

XtIntervalId XtAddTimeOut(*interval, proc, client_data*)
unsigned long *interval*;
XtTimerCallbackProc *proc*;
caddr_t *client_data*;

interval Specifies the time (in milliseconds) interval.

proc Specifies the callback procedure.

client_data Specifies the parameters for the procedure.

XtAddTimeOut creates a timeout value in the default application context and returns an identifier for it. The timeout value is set in *interval*. The callback procedure is called when the time interval elapses. Then, the timeout is removed.

Note: This routine has been replaced by **XtAppAddTimeOut**.

¦ Copyright IBM Corp. 1985, 1991

4.17.16 XtAddWorkProc

XtWorkProcId XtAddWorkProc (*proc, closure*)
XtWorkProc *proc*;
Opaque *closure*;

proc Specifies the work procedure.

closure Specifies the client data for the procedure.

XtAddWorkProc registers a work procedure in the default application context. **XtInitialize** must be called before using this routine.

Note: This routine has been replaced by **XtAppAddWorkProc**.

¦ Copyright IBM Corp. 1985, 1991

4.17.17 XtAlmostProc

```
typedef void(*XtAlmostProc)(Widget, Widget, XtWidgetGeometry*,
                                            XtWidgetGeometry*);
     Widget widget;
     Widget new_widget_return;
     XtWidgetGeometry *request;
     XtWidgetGeometry *reply;
```

widget	Specifies the widget on which the geometry change is requested.
new_widget_return	Specifies the new widget which will store the geometry changes.
request	Specifies the original geometry request sent to the geometry manager.
reply	Specifies the compromise geometry returned by the geometry manager.

XtAlmostProc is a pointer for the procedure type *set_values_almost* for a widget class. Most classes inherit this operation from their superclass by specifying **XtInheritSetValuesAlmost** in the class initialization. The Core *set_values_almost* procedure accepts the compromise.

The *set_values_almost* procedure is called when a client tries to set the widget geometry with **XtSetValues** and the geometry manager cannot satisfy the request. It returns **XtGeometryAlmost** with a compromise geometry.

The *set_values_almost* procedure takes the original geometry and the compromise geometry and determines an acceptable compromise. It returns the compromise in the *new_widget_return* parameter and sends the results to the geometry manager for another try.

¦ Copyright IBM Corp. 1985, 1991

4.17.18 XtAppAddActions

void XtAppAddActions(*application_context, actions, num_actions*);
XtAppContext *application_context*;
XtActionList *actions*;
Cardinal *num_actions*;

application_context Specifies the application context.

actions Specifies the action table to register.

num_args Specifies the number of entries in this action table.

XtAppAddActions declares an action table and registers it with the translation manager. If more than one action is registered with the same name, the action registered most recently is used. If duplicate actions exist in an action table, the first action registered is used. The Intrinsics register an action table for **MenuPopup** and **MenuPopdown** as part of X Toolkit initialization.

¦ Copyright IBM Corp. 1985, 1991

4.17.19 XtAppAddConverter

void XtAppAddConverter(*application_context, from_type, to_type, converter, cor*
num_arg)
XtAppContext *application_context*;
String *from_type*;
String *to_type*;
XtConverter *converter*;
XtConvertArgList *convert_args*;
Cardinal *num_args*;

application_context Specifies the application context.

from_type Specifies the source.

to_type Specifies the destination.

converter Specifies the type converter procedure.

convert_args Specifies how to compute the additional arguments to the converter.

num_args Specifies the number of additional arguments to the converter.

XtAppAddConverter registers a new converter. If the same *from_type* and *to_type* are specified in two calls to **XtAppAddConverter**, the second call overrides the first.

If the converters do not require additional arguments, *convert_args* is **NULL** and *num_args* is zero.

For converters that require additional arguments, use **XtAddressMode** and the structure **XtConvertArgRec** to specify how each argument is derived. (See <**X11/Convert.h**>.)

¦ Copyright IBM Corp. 1985, 1991

4.17.20 XtAppAddInput

XtInputID XtAppAddInput (*application_context, source, condition, proc, client_*
XtAppContext *application_context*;
int *source*;
caddr_t *condition*;
XtInputCallbackProc *proc*;
caddr_t *client_data*

application_context Specifies the application context that identifies the application.

source Specifies the source file descriptor.

condition Specifies the mask that indicates a read, write, or exception condition or another operating system dependent condition.

proc Specifies the callback procedure.

client_data Specifies the argument for the specified procedure.

XtAppAddInput registers a new file as an input source for a specified application. (In this instance, "file" represents any sink or source data.) **XtAppAddInput** also specifies the conditions under which the source can generate events. When input is pending on this source, the callback procedure is called.

The *condition* is some union of **XtInputReadMask**, **XtInputWriteMask**, and **XtInputExceptMask**.

¦ Copyright IBM Corp. 1985, 1991

4.17.21 XtAppAddTimeOut

XtIntervalId XtAppAddTimeOut(*application_context, interval, proc, client_data*)
XtAppContext *application_context*;
unsigned long *interval*;
XtTimerCallbackProc *proc*;
caddr_t *client_data*;

application_context Specifies the application context.

interval Specifies the time (in milliseconds) interval.

proc Specifies the callback procedure.

client_data Specifies the argument for the specified procedure.

XtAppAddTimeOut creates a timeout value and returns an identifier for it. The timeout is set in *interval*.

The callback procedure is called when the time interval elapses. Then, the timeout is removed.

¦ Copyright IBM Corp. 1985, 1991

4.17.22 XtAppAddWorkProc

XtWorkProcId XtAppAddWorkProc (*application_context, client_data*)
XtAppContext *application_context*;
XtWorkProc *proc*;
caddr_t *client_data*

application_context Specifies the application context that identifies the application.

proc Specifies the work procedure.

client_data Specifies the argument for the specified procedure.

XtAppAddWorkProc registers a work procedure for a specified application. While multiple work procedures can be registered, the most recently added procedure is the procedure that is called. However, if a work procedure adds another work procedure, the current work procedure has priority over the most recently added procedure.

XtWorkProcId is an opaque identifier for this work procedure.

¦ Copyright IBM Corp. 1985, 1991

4.17.23 XtAppCreateShell

Widget XtAppCreateShell(*application_name, application_class, widget_class, dis*
args, num_args)

```
String application_name;
String application_class;
WidgetClass widget_class;
Display *display;
ArgList args;
Cardinal num_args;
```

application_name	Specifies the application name. If this parameter is **NULL**, the application name passed to **XtDisplayInitialize** is used.
application_class	Specifies the application class.
widget_class	Specifies the widget class for the application top-level widget, which is usually **applicationShellWidgetClass**.
display	Specifies the display providing the resources.
args	Specifies the argument list for the WM_COMMAND property.
num_args	Specifies the number of arguments in the argument list.

XtAppCreateShell creates a top-level widget that is the root of a widget tree. This routine saves the specified application name and class for qualifying all widget resource specifiers. The application name and class are used as the left-most components in all widget resource names for this application.

XtAppCreateShell creates a new logical application within a program by allowing the specification of a new root in the resource hierarchy. Or, **XtAppCreateShell** creates a shell on another display by using the resource database associated with the other display.

The widget returned by **XtAppCreateShell** has the **WM_COMMAND** property set for session managers.

¦ Copyright IBM Corp. 1985, 1991

4.17.24 XtAppError

void XtAppError(*application_context, message*)
XtAppContext *application_context*;
String *message*;

application_context Specifies the application context.

message Specifies the message to be reported.

XtAppError calls the installed fatal error procedure. To customize and internalize error messages for most application programs, use **XtAppErrorMsg**.

¦ Copyright IBM Corp. 1985, 1991

4.17.25 XtAppErrorMsg

void XtAppErrorMsg(*application_context, name, type, class, default, params, num_params*)
XtAppContext *application_context*;
String *name*;
String *type*;
String *class*;
String *default*;
String **params*;
Cardinal **num_params*;

application_context Specifies the application context.

name Specifies the general kind of error.

type Specifies the detailed name of the error.

class Specifies the resource class.

default Specifies a default message if an error database entry is not found.

params Specifies a pointer to a list of values to store in the message.

num_params Specifies the number of values in the parameter list.

XtAppErrorMsg calls the high-level error handler. The Intrinsics internal errors have *class* set to **XtToolkitError**.

¦ Copyright IBM Corp. 1985, 1991

4.17.26 XtAppGetErrorDatabase

XrmDatabase ***XtAppGetErrorDatabase** (*application_context*)
XtAppContext *application_context*;

application_context Specifies the application context.

XtAppGetErrorDatabase obtains the error database and merges it with an application or widget-specific database with the first call to **XtAppGetErrorDatbaseText**. Then, **XtAppGetErrorDatabase** returns the address of the error database.

¦ Copyright IBM Corp. 1985, 1991

4.17.27 XtAppGetErrorDatabaseText

void XtAppGetErrorDatabaseText(*application_context, name, type, class, default*
buffer_return, nbytes, database)
XtAppContext *application_context*;
char **name*,**type*,**class*;
char **default*;
char **buffer_return*;
int *nbytes*;
XrmDatabase *database*;

application_context Specifies the application context.

name, type Specifies the name and type concatenated to form the resource name of the error message.

class Specifies the resource class of the error message.

default Specifies the default message if an error database entry is not found.

buffer_return Specifies the buffer that will store the error message.

nbytes Specifies the size (in bytes) of the buffer.

database Specifies the name of the alternative database to use. If the application database is to be used, it specifies **NULL**.

XtAppGetErrorDatabaseText obtains the error database text for an error or warning for an error message handler. It returns the appropriate or default message from the error database.

¦ Copyright IBM Corp. 1985, 1991

4.17.28 XtAppGetSelectionTimeout

unsigned long XtAppGetSelectionTimeout (*application_context*);
     XtAppContext *application_context*;

application_context Specifies the application context.

XtAppGetSelectionTimeout gets and returns the current selection timeout (in milliseconds) value. The selection timeout is the time during which two applications (that are communicating) must respond to one another.

The initial timeout value is set by the **selectionTimeout** application resource. Otherwise, the timeout value defaults to five seconds.

¦ Copyright IBM Corp. 1985, 1991

4.17.29 XtAppMainLoop

void XtAppMainLoop(*application_context*)
XtAppContext *application_context;*

application_context Specifies the application context that identifies the application.

XtAppMainLoop processes input from a specified application in the following manner:

It reads the next incoming X event by calling **XtAppNextEvent**.
Then, it dispatches the event to the appropriate registered procedur by calling **XtDispatchEvent**.

This process constitutes the main loop of X Toolkit applications.

XtAppMainLoop does not return a value, because applications are expected to exit in response to user action.

¦ Copyright IBM Corp. 1985, 1991

4.17.30 XtAppNextEvent

void XtAppNextEvent(*application_context, event_return*)
XtAppContext *application_context;*
XEvent **event_return;*

application_context Specifies the application context that identifies the application.

event_return Returns the event information to the specified event structure.

XtAppNextEvent returns the value from the top of a specified application input queue.

If there is no input in the X input queue, **XtAppNextEvent** does the following:

Flushes the X output buffe
Waits for an even
Looks at other input sources and timeout value
Calls any callback procedures triggered by these events

During the time that your application is waiting for input, you can register an idle-time work procedure with **XtWorkProc** for background processing.

¦ Copyright IBM Corp. 1985, 1991

4.17.31 XtAppPeekEvent

Boolean XtAppPeekEvent(*application_context, event_return*)
XtAppContext *application_context*;
XEvent **event_return*;

application_context Specifies the application context that identifies the application.

event_return Returns the event information to the specified event structure.

XtAppPeekEvent returns the value from the top of a specified application input queue without removing the input from the queue. **XtAppPeekEvent** does the following:

If there is an event in the queue, it fills in the event and returns nonzero value.
If there is no event in the queue, it flushes the output buffer an blocks until input is available. (This can be done by calling timeout callback procedures in the process.)
If the input is an event, it fills in the event and returns a nonzer value.
If the input is for an alternate input source, it returns a zero

During the time that your application is waiting for input, you can register an idle-time work procedure with **XtWorkProc** for background processing.

¦ Copyright IBM Corp. 1985, 1991

4.17.32 XtAppPending

XtInputMask XtAppPending (*application_context*)
XtAppContext *application_context*;

application_context Specifies the application context that identifies the application to check.

XtAppPending determines if the input queue has any events for a specified application.

If the queue has events or input sources pending for the X Server, **XtAppPending** returns a nonzero value which is a bitmask OR of **XtIMXEvent**, **XtIMTimer**, and **XtIMAlternate**. (See **XtAppProcessEvent**.)

If the queue has no events pending, **XtAppPending** flushes the output buffer and returns a zero.

¦ Copyright IBM Corp. 1985, 1991

4.17.33 XtAppProcessEvent

void XtAppProcessEvent(*application_context, mask*)
XtAppContext *application_context*;
XtInput *mask*;

application_context Specifies the application context that identifies the application to process.

mask Specifies the types of events to process.

XtAppProcessEvent processes applications that require direct control of the processing for different types of input. This routine processes one time, alternate input, or X events.

XtAppProcessEvent processes:

Timer events by calling the appropriate timer callbacks
Alternate input by calling the appropriate input callbacks
And, X events by calling **XtDispatchEvent**.

If there are no appropriate applications to process, **XtAppProcessEvent** blocks until an applications becomes available. It processes the applications in a random order.

The *mask* is the bitwise inclusive OR of any combination of **XtIMEvent**, **XtIMTimer**, and **XtMAlternateInput**. **XtIMAll** can be used as the bitwise inclusive OR of all event types.

This routine is not called by client applications. See **XtAppMainLoop**.

¦ Copyright IBM Corp. 1985, 1991

4.17.34 XtAppSetErrorHandler

void XtAppSetErrorHandler(*application_context, handler*)
XtAppContext *application_context*;
XtErrorHandler *handler*;

application_context Specifies the application context.

handler Specifies the new fatal error procedure.

XtAppSetErrorHandler registers a procedure to call on fatal error conditions. The default error handler provided by the Intrinsics is **_XtError**. It prints the message to standard error and terminates the application.

Fatal error message handlers should not return. If a fatal error message handler returns, subsequent X Toolkit behavior is undefined.

¦ Copyright IBM Corp. 1985, 1991

4.17.35 XtAppSetErrorMsgHandler

void XtAppSetErrorMsgHandler(*application_context, msg_handler*)
XtAppContext *application_context*;
XtErrorMsgHandler *msg_handler*;

application_context Specifies the application context.

msg_handler Specifies the new fatal error procedure.

XtAppSetErrorMsgHandler registers a procedure to call on fatal error conditions. The default error handler provided by the Intrinsics constructs a string from the error resource database and calls **XtError**.

Fatal error message handlers should not return. If a fatal error message handler returns, subsequent X Toolkit behavior is undefined.

¦ Copyright IBM Corp. 1985, 1991

4.17.36 XtAppSetSelectionTimeout

void XtAppSetSelectionTimeout(*application_context, timeout*)
XtAppContext *application_context*;
unsigned long *timeout*;

application_context Specifies the application context.

timeout Specifies the selection timeout in milliseconds.

XtAppSetSelectionTimeout sets the Intrinsics selection timeout value. If **selectionTimeout** is not specified, it defaults to five seconds.

¦ Copyright IBM Corp. 1985, 1991

4.17.37 XtAppSetWarningHandler

void XtAppSetWarningHandler(*application_context, handler*)
XtAppContext *application_context*;
XtErrorHandler *handler*;

application_context Specifies the application context.

handler Specifies the new nonfatal error procedure.

XtAppSetWarningHandler registers a procedure to call on nonfatal error conditions. The default warning handler provided by the Intrinsics is **_XtWarning**. It prints the message to standard error and returns to the caller.

¦ Copyright IBM Corp. 1985, 1991

4.17.38 XtAppSetWarningMsgHandler

void XtAppSetWarningMsgHandler(*application_context, msg_handler*)
XtAppContext *application_context*;
XtErrorMsgHandler *msg_handler*;

application_context Specifies the application context.

msg_handler Specifies the new nonfatal error procedure.

XtAppSetWarningHandler registers a procedure to call on nonfatal error conditions. The default warning handler provided by the Intrinsics constructs a string from the error resource database. Then, it calls **_XtWarning** and returns to the caller.

¦ Copyright IBM Corp. 1985, 1991

4.17.39 XtAppWarning

void XtAppWarning(*application_context, message*)
XtAppContext *application_context*;
String *message*;

application_context Specifies the application context.

message Specifies the nonfatal error message.

XtAppWarning calls the installed nonfatal error procedure. To customize snd internalize warning messages, use **XtAppWarningMsg**.

¦ Copyright IBM Corp. 1985, 1991

4.17.40 XtAppWarningMsg

void XtAppWarningMsg(*application_context, name, type, class, default, params, num_params*)
XtAppContext *application_context*;
String *name*;
String *type*;
String *class*;
String *default*;
String **params*;
Cardinal **num_params*;

application_context Specifies the application context.

name Specifies the general kind of error.

type Specifies the detailed name of the error.

class Specifies the resource class.

default Specifies a default message if there is no entry in the error database.

params Specifies a pointer to a list of values stored in the message.

num_params Specifies the number of values in the parameter list.

XtAppWarningMsg calls the installed high-level warning handler. Use this routine to customize and internalize warning messages.

The Intrinsics internal warnings have *class* set to **XtToolkitError**.

¦ Copyright IBM Corp. 1985, 1991

4.17.41 XtArgsFunc

```
typedef Boolean(*XtArgsFunc)(Widget, Arglist, Cardinal*);
      Widget widget;
      ArgList args;
      Cardinal *num_args;
```

widget Specifies the target widget.

args Specifies the argument list for **XtCreateWidget**.

num_args Specifies the number of arguments in the argument list.

XtArgsFunc is a pointer for the *set_values_hook* procedure in a widget class. Widgets with a subpart can set the resource values with **XtSetValues** and a *set_values_hook* procedure.

¦ Copyright IBM Corp. 1985, 1991

4.17.42 XtArgsProc

```
typedef void (*XtArgsProc)(Widget, ArgList, Cardinal*);
    Widget widget;
    ArgList args;
    Cardinal *num_args;
```

widget Specifies the widget.

args Specifies the argument list to override the resource defaults.

num_args Specifies the number of arguments in the argument list.

XtArgsProc is a pointer for the *initialize_hook* procedure. This procedure is called:

Immediately after the corresponding initialize if it is not **NULL**.
Instead of the corresponding initialize procedure if it is **NULL**.

¦ Copyright IBM Corp. 1985, 1991

4.17.43 XtArgVal

```
typedef type XtArgVal;

typedef struct {
       String name;
       XtArgVal value;
} Arg, *ArgList;
```

type Specifies a C language type large enough to contain a pointer to a function or a parameter of type **caddr_t**, **char ***, **long**, or **int ***.

name Specifies the name of the resource.

value Contains the resource value if the size of the resource is less than, or equal to, the size of an **XtArgVal**. the resource value is stored directly in *value*. Otherwise, *value* is a pointer to the resource value.

Many of the Intrinsics routines need to be passed pairs of resource names and values, called an ***argument list***. These are passed as an **ArgList** structure.

¦ Copyright IBM Corp. 1985, 1991

4.17.44 XtAugmentTranslations

void XtAugmentTranslations(*widget, translations*)
Widget *widget*;
XtTranslations *translations*;

widget Specifies the widget in which to merge the new translations.

translations Specifies the compiled translation table in which to merge the new translations. This argument cannot **NULL**.

XtAugmentTranslations merges new translations into an existing widget translation table. The new translation is ignored if it contains an event or event sequence that is already in the widget translation table.

¦ Copyright IBM Corp. 1985, 1991

4.17.45 XtBuildEventMask

EventMask XtBuildEventMask(*widget*)
Widget *widget*;

widget Specifies the widget.

XtBuildEventMask retrieves the event mask for a specified widget. It returns the event mask representing the logical OR of all event masks for event handlers registered on the widget with **XtAddEventHandler**. It also returns all event translations, including accelerators, installed on the widget.

¦ Copyright IBM Corp. 1985, 1991

4.17.46 XtCallAcceptFocus

Boolean XtCallAcceptFocus(*widget, time*)
Widget *widget*;
Time **time*;

widget Specifies the widget.

time Specifies the X time of the event causing the accept focus.

XtCallAcceptFocus calls the *accept_focus* procedure for the specified widget. It passes the specified widget and time. And, it returns the information in the *accept_focus* procedure.

XtCallAcceptFocus returns **False** if the *accept_focus* procedure is **NULL**.

¦ Copyright IBM Corp. 1985, 1991

4.17.47 XtCallbackExclusive

void XtCallbackExclusive(*widget, client_data, call_data*)
Widget *widget*;
caddr_t *client_data*;
caddr_t *call_data*;

widget Specifies the widget executing the callback.

client_data Specifies the shell to be popped up.

call_data Specifies the callback data. (This parameter is not used by this procedure.)

XtCallbackExclusive maps a pop-up from a specified widget callback list in the following manner:

It calls **XtPopup** with the shell specified by the *client_data* parameter and the *grab_kind* set to **XtGrabExclusive**.
Then, it sets the widget to be insensitive by calling **XtSetSensitive**.

XtCallbackExclusive is not required in callbacks. In particular, an application must provide customized code for callbacks that create pop-up shells dynamically or callbacks that must do more than desensitize the button.

¦ Copyright IBM Corp. 1985, 1991

4.17.48 XtCallbackList

```
typedef struct{
     XtCallbackProc(callback);
     caddr_t (client_data);
} XtCallbackRec, *XtCallbackList;
```

XtCallbackList specifies the address of a null-terminated array. This structure is used in **XtCreateWidget**, **XtSetValues**, or **XtGetValues** call.

¦ Copyright IBM Corp. 1985, 1991

4.17.49 XtCallbackNone

void XtCallbackNone(*widget, client_data, call_data*)
Widget *widget*;
caddr_t *client_data*;
caddr_t *call_data*;

widget Specifies the widget executing the callback.

client_data Specifies the shell to be popped up.

call_data Specifies the callback data. (This parameter is not used by this procedure.)

XtCallbackNone maps a pop-up from a specified widget callback list in the following manner:

It calls **XtPopup** with the shell specified by the *client_data* parameter and *grab_kind* set to **XtGrabNone**.
Then, it sets the widget to be insensitive by calling **XtSetSensitive**.

XtCallbackNone is not required in callbacks. In particular, an application must provide customized code for callbacks that create pop-up shells dynamically or callbacks that must do more than desensitize the button.

¦ Copyright IBM Corp. 1985, 1991

4.17.50 XtCallbackNonexclusive

void XtCallbackNonexclusive(*widget, client_data, call_data*)
Widget *widget*;
caddr_t *client_data*;
caddr_t *call_data*;

widget Specifies the widget executing the callback.

client_data Specifies the shell to be popped up.

call_data Specifies the callback data. (This parameter is not used by this procedure.)

XtCallbackNonexclusive maps a pop-up from a specified widget callback list in the following manner:

It calls **XtPopup** with the shell specified by the *client_data* parameter and *grab_kind* set to **XtGrabNonexclusive**.
Then, it sets the widget to be insensitive by using **XtSetSensitive**.

XtCallbackNonexclusive is not required in callbacks. In particular, an application must provide customized code for callbacks that create pop-up shells dynamically or callbacks that must do more than desensitize the button.

¦ Copyright IBM Corp. 1985, 1991

4.17.51 XtCallbackPopdown

void XtCallbackPopdown(*widget, client_data, call_data*)
Widget *widget*;
caddr_t *client_data*;
caddr_t *call_data*;

widget Specifies the widget.

client_data Specifies a pointer to the **XtPopdownID** structure.

call_data Specifies the callback data. (This parameter is not used by this procedure.)

XtCallbackPopdown pops down a shell that was mapped by **XtCallbackNone**, **XtCallbackNonexclusive**, or **XtCallbackExclusive**.

The **XtCallbackPopdown** casts the *client_data* parameter to an **XtPopdownID** pointer:

```
typedef struct {
     Widget shell_widget;
     Widget enable_widget;
} XtPopdownIDRec, *XtPopdownID;
```

shell_widget Specifies the shell to popdown.

enable_widget Specifies the widget used to pop the shell up.

XtCallbackPopdown does the following:

It calls **XtPopdown** with the specified *shell_widget*.
Then, it calls **XtSetSensitive** to resensitize the *enable_widget*.

¦ Copyright IBM Corp. 1985, 1991

4.17.52 XtCallbackProc

```
typedef void (*XtCallbackProc)(Widget, caddr_t, caddr_t);
      Widget widget;
      caddr_t client_data;
      caddr_t call_data;
```

widget Specifies the widget for the callback procedure.

client_data Specifies the data that should be passed to the client.

call_data Specifies any callback data that should be passed to the client.

XtCallbackProc is the callback procedures type for callback lists. A client can register the callback for client-specific data, such as a pointer to additional information about the widget in the *client_data* variable. If the necessary information is in the widget, *client_data* is **NULL**.

The *call_data* allows the client to specify data about the callback. This variable helps the client avoid using **XtGetValues** or a widget-specific function to retrieve data from the widget. For example, when Scrollbar executes *thumbChanged* callback list, *call_data* passes the new position of the thumb.

Generally, avoid putting complex state information in the *call_data* argument of the widget. Use the more general data retrieval methods, if necessary.

¦ Copyright IBM Corp. 1985, 1991

4.17.53 XtCallCallbacks

void XtCallCallbacks(*widget, callback_name, call_data*)
Widget *widget*;
String *callback_name*;
caddr_t *call_data*;

widget Specifies the widget.

callback_name Specifies the callback list to be executed.

call_data Specifies callback list with specific data value for the specified callback procedures.

XtCallCallbacks executes the callback procedures in a widget callback list.

The *call_data* can have any of the following values:

The actual data if one 32-bit longword is needed
The address of the data if more than one longword is needed
NULL if no data is required. (For example, the **commandActivated** callback list in *Command* has to notify its clients that the button has been activated.

¦ Copyright IBM Corp. 1985, 1991

4.17.54 XtCalloc

char *XtCalloc(*num, size*)
Cardinal *num*;
Cardinal *size*;

num Specifies the number of array elements to allocate.

size Specifies the size (in bytes) of an array element.

XtCalloc allocates and initializes an array. It allocates space for the specified number of array elements of the specified size and initializes the space to zero.

If there is insufficient memory to allocate the new block, it calls **XtErrorMsg**.

¦ Copyright IBM Corp. 1985, 1991

4.17.55 XtCaseProc

```
typedef void(*XtCaseProc)(KeySym*, KeySym*, KeySym*);
      KeySym *keysym;
      KeySym *lower_return;
      KaySym *upper_return;
```

keysym Specifies the KeySym for conversion.

lower_return Specifies the lowercase equivalent for the KeySym.

upper_return Specifies the uppercase equivalent for the KeySym.

XtCaseProc is a pointer to case converter procedures. It allows capitalization of nonstandard KeySyms with case conversion routines, such as **XtRegisterCaseConverter** and **XtConvertCase**.

If there is no case distinction, **XtCaseProc** stores the KeySym in both return values.

¦ Copyright IBM Corp. 1985, 1991

4.17.56 XtCheckSubclass

void XtCheckSubclass(*widget, widget_class, message*)
Widget *widget*;
WidgetClass *widget_class*;
String *message*;

widget Specifies the widget to check.

widget_class Specifies the widget class to test against.

message Specifies an error message.

XtCheckSubclass checks the subclass of a widget and generates a debugging error message. It determines if the class of the widget is equal to or a subclass of *widget_class*. The widget can be any number of subclasses down the chain.

If the widget is not a subclass of the *widget_class*, **XtCheckSubclass** constructs the following:

An error message from the supplied messag
The actual class of the widge
The expected class of the widget

Then, it calls **XtErrorMsg**.

Use **XtCheckSubclass** at the entry point of exported routines to ensure that a valid widget class for the exported operation was passed.

XtCheckSubclass is executed only when the widget has been compiled with the compiler symbol **DEBUG** defined. Otherwise, **XtCheckSubclass** is defined as an empty string and does not generate code.

¦ Copyright IBM Corp. 1985, 1991

4.17.57 XtClass

WidgetClass XtClass(*widget*)
Widget *widget*;

widget Specifies the widget.

XtClass obtains the class of a widget and returns a pointer to the widget class structure.

¦ Copyright IBM Corp. 1985, 1991

4.17.58 XtClassProc

```
typedef void (*XtClassProc)();

void ClassProc(widgetClass)
      WidgetClass widgetClass;
```

XtClassProc initializes additional fields that are part of a class record using *class_part_initialization* procedure.

Specify **NULL** in the *class_part_initialize* field for classes that do not define new class fields and do not need extra processing.

¦ Copyright IBM Corp. 1985, 1991

4.17.59 XtCloseDisplay

void XtCloseDisplay (*display*)
Display **display*;

display Specifies the display.

XtCloseDisplay closes a display and removes it from an application context as soon as it is possible to do so. For example, if **XtCloseDisplay** is called from within an event dispatch it does not close the display until the dispatch is complete.

Use **XtCloseDisplay** only when an application will be executing after the display is closed. Otherwise, use **XtDestroyApplicationContext** or exit appropriately.

¦ Copyright IBM Corp. 1985, 1991

4.17.60 XtConfigureWidget

void XtConfigureWidget(*widget, x, y, width, height, border_width*)
Widget *widget*;
Position *x*;
Position *y*;
Dimension *width*;
Dimension *height*;
Dimension *border_width*;

widget	Specifies the widget.
x, y	Specifies *x* and *y* coordinates for the new widget.
width, height	Specifies the new width and height for the widget.
border_width	Specifies the new widget size.

XtConfigureWidget moves and resizes the sibling widget of the child making the geometry request. This routine returns immediately if the specified geometry fields are the same value as the old geometry fields. Otherwise, **XtConfigureWidget** writes the new values into the widget.

If the widget is realized, **XtConfigureWidget** calls **XConfigureWindow** on the window of the widget.

If the new value for *width* or *height* is different from the previous values, **XtConfigureWidget** calls the resize procedure of the widget to notify the widget of the size change.

¦ Copyright IBM Corp. 1985, 1991

4.17.61 XtConvert

void XtConvert(*widget, from_type, from, to_type, to_return*)
Widget *widget*;
String *from_type*;
XrmValuePtr *from*;
String *to_type*;
XrmValuePtr *to_return*;

widget Specifies the widget to use for additional arguments.

from_type Specifies the source.

from Specifies the value to be converted.

to_type Specifies the destination.

to_return Returns the converted value.

XtConvert invokes resource conversions. Conversion routines are used if a resource or widget default value in an application are different than what the user requires.

XtConvert looks up the type converter registered to convert *from_type* to *to_type* and computes additional arguments. Then, it calls **XtDirectConvert**.

¦ Copyright IBM Corp. 1985, 1991

4.17.62 XtConvertCase

```
void XtConvertCase(display, keysym, lower_return, upper_return);
      Display *display;
      KeySym keysym;
      KeySym *lower_return;
      KeySym *upper_return;
```

display Specifies the display that provided the KeySym.

keysym Specifies the KeySym to convert.

lower_return Returns the lowercase equivalent of the KeySym.

upper_return Returns the uppercase equivalent of the KeySym.

XtConvertCase determines upper and lowercase equivalents for a KeySym. It calls the appropriate converter and returns the results. This routine can be used with a user-supplied **XtKeyProc**.

¦ Copyright IBM Corp. 1985, 1991

4.17.63 XtConverter

```
typedef void (*XtConverter)(XrmValue*, Cardinal*, XrmValue*, XrmValue*);
      XrmValue *args;
      Cardinal *num_args;
      XrmValue *from;
      XrmValue *to;
```

args Specifies a list of additional **XrmValue** arguments to the converter if additional context is required to perform the conversion. Otherwise, it specifies **NULL**.

num_args Specifies the number of additional **XrmValue** arguments. Otherwise, it specifies zero.

from Specifies the value to convert.

to Specifies the descriptor to use to return the converted value.

XtConverter is a resource converter procedure pointer. It adds arguments to the converter if more context is required to perform the conversion. (For example, the string-to-font converter needs the screen to perform; the string-to-pixel converter needs both the screen and colormap.) The additional arguments are returned in *num_args*.

Type converters use pointers to **XrmValue** structures for input and output values. These structures are defined in **<X11/Xresource.h>**.

¦ Copyright IBM Corp. 1985, 1991

4.17.64 XtConvertSelectionProc

```
typedef Boolean(*XtConvertSelectionProc)(Widget, Atom*, Atom*, Atom*,
                                         caddr_t*, unsigned long*, int*);
      Widget widget;
      Atom *selection;
      Atom *target;
      Atom *type_return;
      caddr_t *value_return;
      unsigned long *length_return;
      int *format_return;
```

widget Specifies the widget that currently owns this selection.

selection Specifies the atom that describes the type of selection requested, for example, **XA_PRIMARY** or **XA_SECONDARY**.

target Specifies the type of the selection requested, for example, a file name, text, or window.

type_return Specifies a pointer to an atom that will store the converted value of the selection. (For instance, a file name or text could have **XA_STRING** as the *type_return.*)

value_return Specifies a pointer that will store the converted value of the selection.

length_return Specifies a pointer that will store the number of elements in value (each by size indicated in format).

format_return Specifies a pointer that will store the size (in bits) of the data elements of the selection value.

XtConvertSelectionProc is a procedure used with atomic transfers. It is used by the selection owner. **XtConvertSelectionProc** is called by the Intrinsics selection mechanism to get the value of selection as a given type from the current selection owner.

The selection owner is responsible for allocating the storage for the *value_return* variable. If the selection owner provided an **XtSelectionDoneProc** for the selection, the storage is owned by the selection owner. Otherwise, the storage is owned by the Intrinsics selection mechanism.

If the selection owner converts the selection target successfully, **XtConvertSelectionProc** returns **True**. Otherwise, it returns **False** indicating that the values are undefined.

Each **XtConvertSelectionProc** should respond to the target value **TARGETS** by returning a value with a list of the targets that will be used for the selection conversion.

¦ Copyright IBM Corp. 1985, 1991

4.17.65 XtCreateApplicationContext

XtAppContext XtCreateApplicationContext;

XtCreateApplicationContext creates an application context. It returns an application context, which is an opaque type. Each application must have at least one application context.

¦ Copyright IBM Corp. 1985, 1991

4.17.66 XtCreateApplicationShell

Widget XtCreateApplicationShell(*name, widget_class, args, num_args*)
String *name*;
WidgetClass *widget_class*;
ArgList *args*;
Cardinal *num_args*;

name Specifies **NULL** because this parameter is ignored.

widget_class Specifies the widget class pointer for the application Shell widget, usually the **topLevelShellWidgetClass** or a subclass.

args Specifies the argument list to override the resource defaults.

num_args Specifies the number of arguments.

XtCreateApplicationShell creates an application Shell widget by calling **XtAppCreateShell** with the following:

An *application_name* of **NULL**
The *application_class* passed to **XtInitialize**
The default application context created by **XtInitialize**.

Note: This routine has been replaced by **XtAppCreateShell**.

¦ Copyright IBM Corp. 1985, 1991

4.17.67 XtCreateManagedWidget

Widget XtCreateManagedWidget(*name, widget_class, parent, args, num_args*)
String *name*;
WidgetClass *widget_class*;
Widget *parent*;
ArgList *args*;
Cardinal *num_args*;

name Specifies the text name for the created widget.

widget_class Specifies the widget class pointer for the created widget.

parent Specifies the parent widget.

args Specifies the argument list to override the resource defaults.

num_args Specifies the number of arguments in the argument list.

XtCreateManagedWidget creates and manages a child widget in a single procedure. It calls **XtCreateWidget** and **XtManageChild**.

¦ Copyright IBM Corp. 1985, 1991

4.17.68 XtCreatePopupShell

Widget XtCreatePopupShell(*name, widget_class, parent, args, num_args*)
String *name*;
WidgetClass *widget_class*;
Widget *parent*;
ArgList *args*;
Cardinal *num_args*;

name Specifies the text name for the created shell widget.

widget_class Specifies the widget class pointer for the created shell widget.

parent Specifies the parent widget.

args Specifies the argument list to override the resource defaults.

num_args Specifies the number of arguments in the argument list.

XtCreatePopupShell creates a pop-up shell. It ensures that the specified class is a subclass of **Shell**. **XtCreatePopupShell** attaches the shell directly to the pop-ups of the parent instead of attaching the widget to the children list of the parent widget.

A spring-loaded pop-up invoked from a translation table must exist at the time that the translation is invoked, so that the translation manager can find the shell by name.

¦ Copyright IBM Corp. 1985, 1991

4.17.69 XtCreateWidget

Widget XtCreateWidget(*name, widget_class, parent, args, num_args*)
String *name*;
WidgetClass *widget_class*;
Widget *parent*;
ArgList *args*;
Cardinal *num_args*;

name Specifies the resource name for the created widget. This name is used for retrieving resources. This name should not be the same as the name for another widget that is a child of same parent.

widget_class Specifies the widget class pointer for the created widget.

parent Specifies the parent widget.

args Specifies the argument list to override the resource defaults.

num_args Specifies the number of arguments in the argument list.

XtCreateWidget creates an instance of a widget. The type for this procedure is **XtWidgetProc**.

XtCreateWidget performs many of the boilerplate operations of widget creation, such as:

Checks if *class_initialize* has been called for this class and for all superclasses. If not, it calls those necessary in a superclass-to-subclass order.

Allocates memory for the widget instance

Allocates memory for the constraints of the parent and stores th address of this memory into the constraints field, if the parent is a subclass of *constraintWidgetClass*.

Initializes the Core non-resource data fields, such as *parent* and *visible*.

Initializes the resource fields, such as *background_pixel*, by using the resource lists specified for this class and all superclasses.

Initializes the resource fields of the constraints record by using th constraint resource list specified for the class of the parent and all superclasses up to *constraintWidgetClass*, if the parent is a subclass of *constraintWidgetClass*.

Calls the initialize procedures for the widget, starting at the Cor initialize procedure and moving down to the widget's initialize procedure.

Puts the widget into its parent's children list by calling th *insert_child* procedure of the parent. See "Adding Children to a Composite Widget" in topic 4.8.2.

Calls the constraint initialize procedures, starting wit *constraintWidgetClass* and moving down to the initialize procedure of

¦ Copyright IBM Corp. 1985, 1991

the parent's constraint, if the parent is a subclass of *constraintWidgetClass*.

The number of arguments in an argument list can be computed automatically with the **XtNumber** macro. See **XtNumber** in this chapter.

¦ Copyright IBM Corp. 1985, 1991

4.17.70 XtCreateWindow

void XtCreateWindow(*widget, window_class, visual, value_mask, attributes*)
Widget *widget*;
unsigned int *window_class*;
Visual **visual*;
XtValueMask *value_mask*;
XSetWindowAttributes **attributes*;

widget Specifies the widget used to create the window.

window_class Specifies the **Xlib** window class which can be **InputOutput**, **InputOnly**, or **CopyFromParent**.

visual Specifies the visual type, which is usually **CopyFromParent**.

value_mask Specifies which attribute fields to use.

attributes Specifies the window attributes for **XCreateWindow**.

XtCreateWindow calls **XCreateWindow** with the widget structure and parameters. Then, it assigns the created window into the window field of the widget.

A realize procedure should call the X-Windows Toolkit analog **XtCreateWindow** rather than call the **Xlib XCreateWindow** routine directly.

¦ Copyright IBM Corp. 1985, 1991

4.17.71 XtDatabase

XrmDatabase XtDatabase (*display*)
Display **display*;

display Specifies the display.

XtDatabase obtains the resource database for a particular display. This routine returns the fully merged resource database built by **XtDisplayInitialize**. If the display returned was not initialized by **XtDisplayInitialize**, the results are not defined.

¦ Copyright IBM Corp. 1985, 1991

4.17.72 XtDestroyApplicationContext

void XtDestroyApplicationContext *(application_context)*
XtAppContext *application_context;*

application_context Specifies the application context.

XtDestroyApplicationContext destroys an application context as soon as it is safe to do so. For example, if this routine is called from within an event dispatch, such as a callback procedure, it does not destroy the application context until the dispatch is completed.
XtDestroyApplicationContext also closes any displays associated with the specified application context.

¦ Copyright IBM Corp. 1985, 1991

4.17.73 XtDestroyGC

void XtDestroyGC(*gc*)
Widget *widget*;
GC *gc*;

widget Specifies the widget.

gc Specifies the graphics context to be deallocated.

XtDestroyGC deallocates a graphics context when it is no longer needed. It counts references to sharable graphics contexts and generates a free request to the server when a graphics context is destroyed.

Note: The release 2 version of **XtDestroyGC** did not require a widget argument. Therefore, this function is not very portable, and you are encouraged to use **XtReleaseGC** instead.

¦ Copyright IBM Corp. 1985, 1991

4.17.74 XtDestroyWidget

void XtDestroyWidget(*widget*)
Widget *widget*;

widget Specifies the widget.

XtDestroyWidget destroys a widget instance. This routine provides the only method for destroying widgets, including widgets that need to destroy themselves. It is used at any time, even from an application callback procedure of the widget being destroyed. The destroy process consists of two phases to avoid dangling references to destroyed widgets.

In Phase 1, **XtDestroyWidget**:

Returns immediately, if *being_destroyed* of the widget is **True**.

Recursively descends the widget tree and sets the *being_destroyed* field to **True** for the widget and all descendants.

Adds the widget to a list of widgets (the destroy list) that should b destroyed when it is safe to do so. Entries on the destroy list satisfy the invariant:

If *w1* occurs before *w2* on the destroy list, then *w2* is not a descendant of *w1*. (The term *descendant* here refers to both normal and pop-up children.)

Phase 2 occurs when all procedures that should execute as a result of the current event have been called, including all procedures registered with the **Event** and **Translation** managers. That is, Phase 2 starts immediately when not in **XtDispatchEvent** or when the current invocation of **XtDispatchEvent**. is about to return.

Phase 2 performs the following actions on each entry in the destroy list:

It calls the destroy callbacks registered on the widget and al descendants in post-order, calling children callbacks before parent callbacks.

If the parent of the widget is a subclass of *compositeWidgetClass* and the parent is not being destroyed, it calls the **XtUnmanageChild** routine on the widget, then calls the *delete_child* procedure of the widget parent. See "Deleting Children of Composite Widgets" in topic 4.8.4.

If the parent of the widget is a subclass of *constraintWidgetClass*, it calls:

1. The constraint destroy procedure for the parent.
2. The constraint destroy procedure for the superclass of the parent.
3. The constraint destroy procedure for *constraintWidgetClass*

Calls the destroy class procedures for the widget and all descendant in post-order:

1. The destroy procedure declared in the widget class
2. The destroy procedure declared in its superclass
3. The destroy procedure declared in the Core widget class record.

¦ Copyright IBM Corp. 1985, 1991

Calls **XDestroyWindow** if the widget is realized or if it has a window. The server recursively destroys all descendant windows.

Finally, **XtDestroyWidget** recursively descends the tree and deallocates all pop-up widgets, constraint records, callback lists, and, if the widget is a subclass of **compositeWidgetClass**, all children.

¦ Copyright IBM Corp. 1985, 1991

4.17.75 XtDirectConvert

void XtDirectConvert(*converter, args, num_args, from, to_return*)
    XtConverter *converter*;
    XrmValuePtr *args*;
    Cardinal *num_args*;
    XrmValuePtr *from*;
    XrmValuePtr *to_return*;

converter Specifies the conversion procedure to call.

args Specifies the argument list for the conversion.

num_args Specifies the number of additional arguments.

from Specifies the value to be converted.

to_return Returns the converted value.

XtDirectConvert invokes resource conversions. It looks in the converter cache to see if this conversion procedure has been called with the specified arguments. If this conversion procedure has been called, it returns a descriptor for information stored in the cache.

Before calling the specified converter, **XtDirectConvert** sets the return value to zero and the return value address to **NULL**. Then, it calls the converter and enters the results in the cache.

If the *to_return* address contains a value other than **NULL**, the conversion was successful.

XtDirectConvert is usually called after **XtConvert**.

¦ Copyright IBM Corp. 1985, 1991

4.17.76 XtDisownSelection

void XtDisownSelection(*widget, selection, time*)
Widget *widget*;
Atom *selection*;
Time *time*;

widget Specifies the widget that will lose ownership.

selection Specifies the atom that identifies the selection to be lost.

time Specifies the timestamp that indicates when the selection ownership is relinquished.

XtDisownSelection informs the Intrinsics selection mechanism that the specified widget is to lose ownership of the selection.

After **XtDisownSelection** is called, the widget *convert_proc* cannot be called. On the other hand, the widget *done_proc* can be called if a conversion procedure, which was started before the call to **XtDisownSelection**, finishes after this routine is started.

XtDisownSelection takes no action if the specified widget does not currently own the selection.

¦ Copyright IBM Corp. 1985, 1991

4.17.77 XtDispatchEvent

Boolean XtDispatchEvent(*event*)
XEvent **event*;

event Specifies a pointer to the event structure to be dispatched to the appropriate event handler.

XtDispatchEvent receives X events. Then, it calls the appropriate event handlers and passes the widget, the event, and client-specific data registered with each procedure. It sends those events to the event handler functions that have been registered previously with the dispatch routine.

XtDispatchEvent returns the following:

True, if it dispatched the event to an event handler.
False, if it did not find an event handler to receive the event.

XtDispatchEvent dispatches events acquired with the **XtAppNextEvent** routine and events constructed by the user.

If there are no event handlers for the registered events, the event is ignored and the dispatcher returns.

Also, **XtDispatchEvent** implements the grab semantics for **XtAddGrab**.

¦ Copyright IBM Corp. 1985, 1991

4.17.78 XtDisplay

Display *XtDisplay(*widget*)
     Widget *widget*;

XtDisplay returns the display pointer for the specified widget.

¦ Copyright IBM Corp. 1985, 1991

4.17.79 XtDisplayInitialize

void XtDisplayInitialize (*application_context, display, application_name, appl*
options, num_options, argc, argv);

XtAppContext *application_context*;
Display **display*;
String *application_name*;
String *application_class*;
XrmOptionDescRec **options*;
Cardinal *num_options*;
Cardinal **argc*;
String **argv*;

application_context Specifies the application context.

display Specifies the display. Each display can be in one application context only.

application_name Specifies the name of the application instance.

application_class Specifies the class name of this application, which is usually the generic name for all instances of this application.

options Specifies how to parse the command line for any application-specific resources. The *options* argument is passed as a parameter to **XrmParseCommand**.

num_options Specifies the number of entries in the options list.

argc Specifies a pointer to the number of command line parameters.

argv Specifies the command line parameters.

XtDisplayInitialize initializes a display and adds it to an application context. It builds the resource database, calls the **Xlib XrmParseCommand** function to parse the command line, and performs other individual display initialization.

After **XrmParseCommand** has been called, *argc* and *argv* contain only those parameters that were not in the standard option table or in the table specified by the *options* parameter.

If the modified *argc* is not zero, most applications print out the modified *argv* and a message listing the allowable options.

The application name is typically the final component of *argv*[0].

XtDisplayInitialize has a table of standard command line options that are passed to **XrmParseCommand** for adding resources to the resource database. **XtDisplayInitialize** also takes additional application-specific resource abbreviations as a parameter. The format of this table is:

```
typedef enum {

   XrmoptionNoArg,          /* Value specified in              */
                            OptionDescRec.value
   XrmoptionIsArg,          /* Value is the option string      */
```

¦ Copyright IBM Corp. 1985, 1991

```
    XrmoptionStickyArg,             /* Value is characters following   */
                                    option
    XrmoptionSepArg,                /* Value is next argument in       */
                                    argv
    XrmoptionSkipArg,               /*Ignore this option and the       */
                                    next argument in argv
    XrmoptionSkipLine               /*Ignore this option and the       */
                                    rest of argv

} XrmOptionKind;


typedef struct {


    char *option;                   /* Option name in argv             */
    char *specifier;                /* Resource name (without          */
                                    application name)
    XrmOptionKind argKind;          /* Style of option                 */
    caddr_t value;                  /* Value to provide if             */
                                    XrmoptionNoArg

} XrmOptionDescRec, *XrmOptionDescList;
```

The standard **XtDisplayInitialize** command line options are:

Option String	Resource	Argument name	Resource Value
-background	background	SepArg	next argument
-bd	borderColor	SepArg	next argument
-bg	background	SepArg	next argument
-borderwidth	borderWidth	SepArg	next argument
-bordercolor	borderColor	SepArg	next argument
-bw	borderWidth	SepArg	next argument
-display	display	SepArg	next argument
-fg	foreground	SepArg	next argument
-fn	font	SepArg	next argument
-font	font	SepArg	next argument

Copyright IBM Corp. 1985, 1991

-foreground	foreground	SepArg	next argument
-geometry	geometry	SepArg	next argument
-iconic	iconic	NoArg	true
-name	name	SepArg	next argument
-reverse	reverseVideo	NoArg	on
-rv	reverseVideo	NoArg	on
+rv	reverseVideo	NoArg	off
-selectionTimeout	selectionTimeout	SepArg	next argument
-synchronous	synchronize	NoArg	on
+synchronous	synchronize	NoArg	off
-title	title	SepArg	next argument
-xrm	next argument	ResArg	next argument

Notes:

1. Any unique abbreviation for an option name in the standard table or in the application table is accepted.
2. If **XtOpenDisplay** is called before **XtDisplayInitialize**, any **-display** or **-name** options that are specified with **XtOpenDisplay** override the same options specified for **XtDisplayInitialize**.
3. If the reverseVideo resource is **True**, the Intrinsics exchange **XtDefaultForeground** and **XtDefaultBackground** for widgets created on this display. (See "Predefined Resource Converters" in topic 4.15.)
4. If the synchronize resource for the specified application is **True**, **XtDisplayInitialize** calls the **Xlib XSynchronize** function to put **Xlib** into synchronous mode for this display connection.
5. The -xrm option provides a method of setting any resource in an application. The next argument should be a quoted string identical (in format) to a line in the user resources file. For example, to give a red background to all command buttons in an application named **xmh**, you do the following:

```
xmh  -xrm 'xmh*Command.background: red'
```

When **XtDisplayInitialize** fully parses the command line, it merges the application option table with the standard option table. Then, it calls the **XrmParseCommand** routine.

If an entry in the application table is the same as an entry in the

¦ Copyright IBM Corp. 1985, 1991

standard table, it overrides the standard table entry. If an option name is a prefix of another option name, both names are kept in the merged table. (Store option tables by option name.)

¦ Copyright IBM Corp. 1985, 1991

4.17.80 XtError

void XtError(*message*)
String *message*;

message Specifies the error message.

XtError calls the installed fatal error procedure. Use **XtErrorMsg** to customize and internalize error messages for most programs.

¦ Copyright IBM Corp. 1985, 1991

4.17.81 XtErrorHandler

```
typedef void (*XtErrorHandler)(String);
    String message;
```

message Specifies the error message.

XtErrorHandler is the low-level error and warning handler procedure type. The error handler should display the specified message string in an appropriate manner.

¦ Copyright IBM Corp. 1985, 1991

4.17.82 XtErrorMsg

void XtErrorMsg (*name, type, class, default, params, num_params*)
String *name*;
String *type*;
String *class*;
String *default*;
String **params*;
Cardinal **num_params*;

name Specifies the general kind of error.

type Specifies the detailed name of the error.

class Specifies the resource class.

default Specifies a default message if an error database entry is not found.

params Specifies a pointer to a list of values to be stored in the message.

num_params Specifies the number of values in the parameter list.

XtErrorMsg calls the high-level error handler. The Intrinsics internal errors have *class* set to **XtToolkitError**.

¦ Copyright IBM Corp. 1985, 1991

4.17.83 XtErrorMsgHandler

```
typedef void(*XtErrorMsgHandler)(String, String, String, String, String*, Card
      String name;
      String type;
      String class;
      String defaultp;
      String *params;
      Cardinal *num_params;
```

name Specifies the name that is concatenated with the specified type to form the resource name of the error message.

type Specifies the type that is concatenated with the name to form the resource name of the error message.

class Specifies the resource class of the error message.

defaultp Specifies a default message if an error database entry is not found.

params Specifies a pointer to a list of values to be substituted in the message.

num_params Specifies the number of values in the parameter list.

XtErrorMsgHandler is the high-level error and warning handler procedure type.

The specified name can be a general error, such as *invalidParameters* or *invalidWindow*.

The specified type gives extra information

The standard **printf** notation is used to substitute the parameters into the message.

¦ Copyright IBM Corp. 1985, 1991

4.17.84 XtEventHandler

```
typedef void (*XtEventHandler)(Widget, caddr_t, XEvent*);
      Widget widget;
      caddr_t client_data;
      XEvent *event;
```

widget Specifies the widget for which to handle events.

client_data Specifies the client specific information registered with the event handler. If the event handler is registered by the widget, it specifies **NULL**.

event Specifies the triggering event.

XtEventHandler is the event handler procedure type for widgets that must use event handlers explicitly. Most widgets use the translation manager, instead of using event handlers explicitly.

¦ Copyright IBM Corp. 1985, 1991

4.17.85 XtExposeProc

```
typedef void (*XtExposeProc)(Widget, XEvent*, Region);
      Widget widget;
      XEvent *event;
      Region region;
```

widget Specifies the widget requiring redisplay.

event Specifies the exposure event giving the rectangle requiring redisplay.

region Specifies the union of all rectangles in this exposure sequence.

XtExposeProc is the expose procedure type for widget classes. The expose procedure redisplays a widget upon exposure. If a widget has no display semantics, specify **NULL** for the expose field. For example, many composite widgets serve as containers for their children only and have no expose procedure.

Note: If the expose procedure is **NULL**, **XtRealizeWidget** fills in a default bit gravity of **NorthWestGravity** before it calls the widget realize procedure.

If the widget *compress_exposure* is **False**, the *region* will be **NULL**. If the widget *compress_exposure* is **True**, the *event* will contain the bounding box for the *region*. See "Compressing Events Using Event Filters" in topic 4.12.2.

¦ Copyright IBM Corp. 1985, 1991

4.17.86 XtFree

void XtFree(*ptr*)
char **ptr*;

ptr Specifies a pointer to the block of storage to be freed.

XtFree frees an allocated block of storage. It returns the storage and allows the storage to be reused. **XtFree** returns immediately if *ptr* is **NULL**.

¦ Copyright IBM Corp. 1985, 1991

4.17.87 XtGeometryHandler

```
typedef XtGeometryResult(*XtGeometryHandler)(Widget, XtWidgetGeometry*,
                                             XtWidgetGeometry*);
      Widget widget;
      XtWidgetGeometry *request;
      XtWidgetGeometry *geometry_return;
```

XtGeometryHandler is the geometry manager procedure type in a composite widget. A class can inherit the geometry manager of its superclass during class initialization.

XtGeometryHandler is also the *query_geometry* procedure type.

The *query_geometry* procedure:

Examines the bits set in *intended->request_mode*

Evaluates the preferred geometry of the widge

Stores the result in *preferred_return*. It sets the bits in *preferred_return->request_mode* to the corresponding geometry fields.

This procedure can generate anyone of the following return values:

XtGeometryYes if the proposed geometry change is acceptable without modification

XtGeometryAlmost if at least one field in *preferred_return* is different from the corresponding field in *intended* or if a bit is set in *preferred_return* that is not set in *intended*

XtGeometryNo if the preferred geometry is identical to the current geometry.

¦ Copyright IBM Corp. 1985, 1991

4.17.88 XtGetApplicationResources

void XtGetApplicationResources(*widget, base, resources, num_resources, args, r*
Widget *widget*;
caddr_t *base*;
XtRourceList *resources*;
Cardinal *num_resources*;
ArgList *args*;
Cardinal *num_args*;

widget Specifies the widget that identifies the resource database to search. (This is the database associated with the display for this widget.)

base Specifies the base address of the subpart data structure where the resources should be written.

resources Specifies the resource list for the subpart.

num_resources Specifies the number of resources in the resource list.

args Specifies the argument list to override resources obtained from the resource database.

num_args Specifies the number of arguments in the argument list.

XtGetApplicationResources retrieves resources that are not specific to a widget, but apply to the overall application.

XtGetApplicationResources does the following:

It constructs a resource name and class list with the specifie widget, which is usually an application shell.

It retrieves the resources from the argument list, the resourc database, or the resource list default values.

Then, it adds the *base* to each address and copies the resources into the specified address.

If *args* is **NULL**, *num_args* must be zero. However, if *num_args* is zero, *args* is not referenced.

Declare resources as members of a structure and pass the address of the structure as the base argument.

¦ Copyright IBM Corp. 1985, 1991

4.17.89 XtGetErrorDatabase

XrmDatabase *XtGetErrorDatabase()

XtGetErrorDatabase obtains the error database and returns the address of the error database.

The Intrinsics performs a binding of the error database and waits for a call **XtGetErrorDatabaseText**. Then, it merges the error database with the application or widget-specific database.

¦ Copyright IBM Corp. 1985, 1991

4.17.90 XtGetErrorDatabaseText

```
void XtGetErrorDatabaseText(name, type, class, default, buffer_return, nbytes)
     char *name, *type, *class;
     char *default;
     char *buffer_return;
     int *nbytes;
```

name, type Specifies the name and type that are concatenated to form the resource name of the error message.

class Specifies the resource class of the error message.

default Specifies the default message if an entry in the error database is not found.

buffer_return Specifies the buffer into which the error message is to be returned.

nbytes Specifies the size (in bytes) of the buffer.

XtGetErrorDatabaseText obtains the error database text for an error or warning. It returns the appropriate or a default message from the error database.

¦ Copyright IBM Corp. 1985, 1991

4.17.91 XtGetGC

GC XtGetGC(*widget, value_mask, values*)
      Widget *widget*;
      XtGCMask *value_mask*;
      XGCValues **values*;

widget Specifies the widget.

value_mask Specifies the value fields.

values Specifies the values for this **GC**.

XtGetGC returns a read-only sharable **GC**. It shares only the GCs with the same values as the values returned by the **Xlib XCreateGC** function.

XtGetGC uses the *value_mask* to inform the server which fields should be filled in with widget data and which fields should be filled in with default values.

¦ Copyright IBM Corp. 1985, 1991

4.17.92 XtGetResourceList

void XtGetResourceList(*widget_class, resources_return, num_resources_return*);
WidgetClass *widget_class*;
XtResourceList **resources_return*;
Cardinal **num_resources_return*;

widget_class	Specifies the widget class pointer for the created shell widget.
resources_return	Specifies a pointer to the storage place for the resource list returned.
num_resources_return	Specifies a pointer to the storage place for the number of entries in the resource list.

XtGetResourceList obtains the resource list structure for a particular class.

If **XtGetResourceList** is called before the widget class is initialized, it returns the resource list specified in the widget class record.

If **XtGetResourceList** is called after the widget class is initialized, it returns a merged resource list that contains the resources for all superclasses.

After **XtGetResourceList** completes, use **XtFree** to free the allocated storage.

¦ Copyright IBM Corp. 1985, 1991

4.17.93 XtGetSelectionTimeout

unsigned long XtGetSelectionTimeout()

XtGetSelectionTimeout obtains the current selection timeout. The selection timeout is the time during which two applications (which are communicating) must repond to each other. If one application does not respond within this time, the Intrinsics aborts the selection request.

Use **XtSetSelectionTimeout** to specify the selection timeout. If a selection timeout is not specified, it defaults to five seconds.

¦ Copyright IBM Corp. 1985, 1991

4.17.94 XtGetSelectionValue

void XtGetSelectionValue(*widget, selection, target, callback, client_data, tim*
Widget *widget*;
Atom *selection*;
Atom *target*;
XtSelectionCallbackProc *callback*;
caddr_t *client_data*;
Time *time*;

widget Specifies the widget making the request.

selection Specifies the selection desired, for example, primary or secondary.

target Specifies the type of information needed about the selection.

callback Specifies the callback procedure.

client_data Specifies the argument for the specified callback procedure.

time Specifies the timestamp that indicates when the selection will be started. (**CurrentTime** cannot be used.)

XtGetSelectionValue obtains the selection value in a single, logical unit. It requests the value of the selection that has been converted to the target type.

The callback procedure communicates the selection values back to the client. The callback procedure may be called before or after **XtGetSelectionValue** returns.

The timestamp triggers the **XtGetSelectionValue** routine.

¦ Copyright IBM Corp. 1985, 1991

4.17.95 XtGetSelectionValues

void XtGetSelectionValues (*widget, selection, targets, count, callback, client*
Widget *widget*;
Atom *selection*;
Atom **targets*;
int *count*;
XtSelectionCallbackProc *callback*;
caddr_t *client_data*;
Time *time*;

widget Specifies the widget making the request.

selection Specifies the particular selection, such as, primary or secondary.

targets Specifies the types of information (about the selection) that is needed.

count Specifies the length of the targets and client data lists.

callback Specifies the callback procedure.

client_data Specifies the client data (one for each target type) passed to the callback procedure.

time Specifies the time that indicates when the selection will be started. (**CurrentTime** cannot be used.)

XtGetSelectionValues takes a list of target types and client data and obtains the current value of the selection converted to each of the targets.

The callback is called once with the corresponding client data for each target. The callback procedure communicates the selection values to the client.

XtGetSelectionValues guarantees that all the conversions use the same selection value because ownership of the selection cannot change within the list.

The timestamp triggers the **XtGetSelectionValues** routine.

¦ Copyright IBM Corp. 1985, 1991

4.17.96 XtGetSubresources

void XtGetSubresources(*widget, base, name, class, resources, num_resources, ar*
num_args)

```
Widget widget;
caddr_t base;
String name;
String class;
XtResourceList resources;
Cardinal num_resources;
ArgList args;
Cardinal num_args;
```

widget	Specifies the widget that wants resources for a subpart.
base	Specifies the base address of the subpart data structure where the resources should be written.
name	Specifies the name of the subpart.
class	Specifies the class of the subpart.
resources	Specifies the resource list for the subpart.
num_resources	Specifies the number of resources in the resource list.
args	Specifies the argument list to override resources obtained from the resource database.
num_args	Specifies the number of arguments in the argument list.

XtGetSubresources obtains resources other than widgets. It constructs a name and class list from the following:

The application name and application clas
The names and classes of the ancestor
The widget

Then, it appends the name and class pair from **XtGetSubresources** to this list.

The resources are taken from the argument list, the resource database, or the default values in the resource list. Then, the resources are copied into the subpart record.

If *args* is **NULL**, *num_args* must be zero. However, if *num_args* is zero, *args* is not referenced.

¦ Copyright IBM Corp. 1985, 1991

4.17.97 XtGetSubvalues

void XtGetSubvalues(*base, resources, num_resources, args, num_args*)
caddr_t *base*;
XtResourceList *resources*;
Cardinal *num_resources*;
ArgList *args*;
Cardinal *num_args*;

base Specifies the base address of the subpart data structure from which the resources should be retrieved.

resources Specifies the non-widget resources list.

num_resources Specifies the number of resources in the resource list.

args Specifies the argument list of name and address pairs with the resource name and the address into which the resource value is to be stored.

num_args Specifies the number of arguments in the argument list.

XtGetSubvalues retrieves the current value of a non-widget resource data associated with a widget instance. It obtains the resource values from the structure identified by the base.

The arguments and values (passed) are dependent on the subpart of the widget. The storage used for the argument list must be deallocated by the application when it is no longer needed.

For a discussion of non-widget subclass resources, see "XtGetSubresources" in topic 4.17.96.

¦ Copyright IBM Corp. 1985, 1991

4.17.98 XtGetValues

void XtGetValues(*widget, args, num_args*)
Widget *widget*;
ArgList *args*;
Cardinal *num_args*;

widget Specifies the widget.

args Specifies the argument list of name and address pairs.

num_args Specifies the number of arguments in the argument list.

XtGetValues retrieves the current value of a resource associated with a widget instance. It starts with the resources specified for the Core widget fields and proceeds down the subclass chain to the widget. The value field of the argument should contain the resource name and the address in which the resource value will be stored. Allocate and deallocate this storage space according to the size of the resource representation type used within the widget.

If the parent of the widget is a subclass of **constraintWidgetClass**, then **XtGetValues** calls the values for any constraint resources requested. **XtGetValues** starts with the constraint resources specified for **constraintWidgetClass** and proceeds down the subclass chain to the constraint resources of the parent.

If the argument list contains a resource name that is not found in the resource lists, the value at the corresponding address is not modified.

Finally, the *get_values_hook* procedures (that are non-NULL) are called in superclass-to-subclass order after the resource values have been called by **XtGetValues**. This permits a subclass to provide non-widget resource data to **XtGetValues**.

¦ Copyright IBM Corp. 1985, 1991

4.17.99 XtHasCallbacks

```
typedef enum {XtCallbackNoList, XtCallbackHasNone, XtCallbackHasSome}
                                XtCallbackStatus;

XtCallbackStatus XtHasCallbacks(widget, callback_name)
      Widget widget;
      String callback_name;
```

widget Specifies the widget to be checked.

callback_name Specifies the callback list.

XtHasCallbacks finds the status of a specified widget callback list. First, it searches for for a callback list, which is identified by *callback_name*. Then, it returns one of the following:

XtCallbackNoList if the callback list does not exist.
XtCallbackHasNone if the callback list is empty.
XtCallbackHasSome if the callback list contains at least one registered callback procedure.

¦ Copyright IBM Corp. 1985, 1991

4.17.100 XtInitialize

Widget XtInitialize(*shell_name, application_class, options, num_options, argc,*
String *shell_name*;
String *application_class*;
XrmOptionDescRec *options*[ ];
Cardinal *num_*options;
Cardinal **argc*;
String *argv*[ ];

shell_name	Specify **NULL** because this parameter is ignored.
application_class	Specifies the class name of this application.
options	Specifies how to parse the command line for any application-specific resources. This argument is passed as a parameter to **XrmParseCommand**.
num_options	Specifies the number of entries in the options list.
argc	Specifies a pointer to the number of command line parameters.
argv	Specifies the command line parameters.

XtInitialize does the following:

Calls **XtToolkitInitialize** to initialize the toolkit internals.
Creates a default application context
Calls **XtOpenDisplay** with a display_string and **NULL** as the application name.
Calls **XtAppCreateShell** with **NULL** as the application name.
Returns the created shell

If **XtInitialize** is called more than once, the behavior of the toolkit is undefined.

¦ Copyright IBM Corp. 1985, 1991

4.17.101 XtInitProc

```
typedef void (*XtInitProc) (Widget, Widget);
      Widget request;
      Widget new;
```

request Specifies the widget with resource values as requested by the argument list, the resource database, and the widget defaults.

new Specifies a widget with the new values, both resource and non-resource, that are allowed.

XtInitProc is the initialize procedure type for a widget class.

¦ Copyright IBM Corp. 1985, 1991

4.17.102 XtInputCallbackProc

```
typedef void (*XtInputCallbackProc)(caddr_t, int*, XtInputId*);
     caddr_t client_data;
     int source;
     XtInputId id;
```

client_data Specifies the client data registered for this procedure in **XtAppAddInput**.

source Specifies the source file descriptor generating the event.

id Specifies the ID returned from the corresponding **XtAppAddInput** call.

XtInputCallbackProc is the callback procedure invoked when file input is pending.

¦ Copyright IBM Corp. 1985, 1991

4.17.103 XtInstallAccelerators

void XtInstallAccelerators(*destination, source*)
Widget *destination*;
Time *source*;

destination Specifies the widget that will receive the accelerators.

source Specifies the widget that will supply the accelerators.

XtInstallAccelerators installs accelerators from a *source* widget to *destination* widget. It installs the accelerators by augmenting the destination translations with the source accelerators.

If the *source* display_accelerator method is non-NULL, **XtInstallAccelerators** calls it with the *source* widget and a string representation of the accelerator table. This indicates that the accelerators have been installed and should be displayed appropriately.

The string representation of the accelerator table is its canonical translation table representation.

¦ Copyright IBM Corp. 1985, 1991

4.17.104 XtInstallAllAccelerators

void XtInstallAllAccelerators(*destination, source*)
Widget *destination*;
Widget *source*;

destination Specifies the widget to receive the accelerators.

source Specifies the root widget to supply the accelerators.

XtInstallAllAccelerators installs all the *source* accelerators from a widget and all the descendants of the widget onto one *destination* widget. It descends the widget tree rooted at the *source* widget. Then, it installs the accelerators that it encounters onto the *destination* widget.

XtInstallAllAccelerators is used with the application main window as the *source* widget.

¦ Copyright IBM Corp. 1985, 1991

4.17.105 XtIsComposite

Boolean XtIsComposite(*widget*)
     Widget *widget*;

widget Specifies the widget.

XtIsComposite determines whether a specified widget is a subclass of Composite.

XtIsComposite is equivalent to **XtIsSubclass** with **compositeWidgetClass** specified.

¦ Copyright IBM Corp. 1985, 1991

4.17.106 XtIsManaged

Boolean XtIsManaged(*widget*)
Widget *widget*

widget Specifies the widget.

XtIsManaged determines the managed state of a specified child widget. It returns **True** if the child widget is managed. Otherwise, it returns **False**.

¦ Copyright IBM Corp. 1985, 1991

4.17.107 XtIsRealized

Boolean XtIsRealized(*widget*)
     Widget *widget*;

widget Specifies the widget.

XtIsRealized determines if a widget has been realized. It returns **True** if the widget is realized.

¦ Copyright IBM Corp. 1985, 1991

4.17.108 XtIsSensitive

Boolean XtIsSensitive(*widget*)
Widget *widget*;

widget Specifies the widget to be checked.

XtIsSensitive determines the current sensitivity state of a widget. (This routine is usually invoked by the parent widget.) It indicates whether user input events are being dispatched.

When a widget is dormant, it is considered insensitive. Therefore, the event manager does not dispatch the following events:

KeyPress, KeyRelease
ButtonPress, ButtonRelease
MotionNotify, EnterNotify, LeaveNotify
FocusIn, FocusOut

See **XtSetSensitive**.

¦ Copyright IBM Corp. 1985, 1991

4.17.109 XtIsSubclass

Boolean **XtIsSubclass**(*widget, widget_class*)
Widget *widget*;
WidgetClass *widget_class*;

widget Specifies the widget.

widget_class Specifies the widget class to test against.

XtIsSubclass determines the subclass of the widget. If the widget is equal to or is a subclass of the specified *widget_class*, this routine returns **True**. The widget does not need to be an immediate subclass of the specified *widget_class*. It can be further down the subclass chain.

Use **XtIsSubclass** to determine the subclass of composite widgets.

¦ Copyright IBM Corp. 1985, 1991

4.17.110 XtKeyProc

```
typedef void(*XtKeyProc)(Display*, KeyCode, Modifiers*, KeySym*);
      Display *display;
      KeyCode keycode;
      Modifiers modifiers;
      Modifiers *modifiers_return;
      KeySym *keysym_return;
```

display	Specifies the display supplying the Keycode.
keycode	Specifies the KeyCode to translate.
modifiers	Specifies the modifiers to the KeyCode.
modifiers_return	Returns a mask that indicates the subset of all modifiers that are examined by the key translator.
keysym_return	Returns the new KeySym.

XtKeyProc is a KeyCode to KeySym translation procedure. It produces a KeySym with a KeyCode and modifiers. For any key translator function, *modifiers_return* indicates the subset of all modifiers examined by the key translator.

¦ Copyright IBM Corp. 1985, 1991

4.17.111 XtLoseSelectionProc

```
typedef void(*XtLoseSelectionProc)(Widget, Atom*);
      Widget widget;
      Atom *selection;
```

widget Specifies the widget that has lost selection ownership.

selection Specifies the atom that describes the selection type.

XtLoseSelectionProc informs the specified widgets that it has lost ownership of the specified selection. This routine does not ask the widget to lose the selection ownership.

¦ Copyright IBM Corp. 1985, 1991

4.17.112 XtMainLoop

void XtMainLoop();

XtMainLoop processes input. First, it reads the next incoming file, timer, or X-Windows event by calling **XtNextEvent**. Then, it dispatches this event to the appropriate registered procedure by calling **XtDispatchEvent**.

XtMainLoop is an infinite loop and does not return. Applications should exit directly in response to a user action.

Note: This routine has been replaced by **XtAppMainLoop**.

¦ Copyright IBM Corp. 1985, 1991

4.17.113 XtMakeGeometryRequest

XtGeometryResult XtMakeGeometryRequest(*widget, request, reply_return*)
Widget *widget*;
XtWidgetGeometry **request*;
XtWidgetGeometry **reply_return*;

widget Specifies the widget ID of the widget making the request.

request Specifies the desired widget geometry (size, position, border width, and stacking order).

reply_return Returns the allowed widget size.

XtMakeGeometryRequest is a request from the child widget to a parent widget for a geometry change. (If a widget is not interested in handling **XtGeometryAlmost**, *reply_return* should be **NULL**.)

After **XtMakeGeometryRequest** is invoked, the geometry manager returns one of the following: **XtGeometryYes, XtGeometryNo, or XtGeometryAlmost**. **XtMakeGeometryRequest** does not return **XtGeometryDone**.

Depending on the conditions, **XtMakeGeometryRequest** performs the following:

Makes the changes and returns **XtGeometryYes** if the widget is unmanaged or the parent of the widget is not realized.

Issues an error if the parent is not a subclass o **compositeWidgetClass** or the geometry manager of the parent is **NULL**.

Returns **XtGeometryNo** if the widget *being_destroyed* is **True**.

Returns **XtGeometryYes** if the widget *x, y, width, height* and *border_width* fields are equal to the values requested. Otherwise, it calls the geometry manager procedure of the parent with the specified parameters.

Calls **Xlib XConfigureWindow** function to reconfigure the widget window (size, location and stacking order), if the following occur:

- The parent geometry manager returns **XtGeometryYes**
- The *request_mode* is not **XtCWQueryOnly**
- And, if the widget is realized.

Otherwise, **XtMakeGeometryRequest** returns the results from the parent geometry manager.

XtMakeGeometryRequest always returns **XtGeometryYes** when called by a primitive widget because the children of primitive widgets are always unmanaged.

The following defined values for *request_mode* are in **<X11/X.h>**.

```
#define CWX                (1<<0)
#define CWY                (1<<1)
#define CWWidth            (1<<2)
#define CWHeight           (1<<3)
#define CWBorderWidth      (1<<4)
#define CWSibling          (1<<5)
#define CWStackMode        (1<<6)
```

¦ Copyright IBM Corp. 1985, 1991

The Intrinsics also supports:

```
#define XtCWQueryOnly       (1<<7)
```

XtCWQueryOnly indicates that the corresponding geometry request is only a query and no widgets should be changed.

The following defined values for *stack_mode* are in **<X11/X.h>**.

```
#define Above           0
#define Below           1
#define TopIf           2
#define BottomIf        3
#define Opposite        4
```

The Intrinsics also supports:

```
#define XtSMDontChange        5
```

XtSMDontChange indicates that the widget requires its current stacking order be preserved.

For the definition and behavior of the *stack_mode,* see Chapter 1, "Using X-Windows" in topic 1.0.

¦ Copyright IBM Corp. 1985, 1991

4.17.114 XtMakeResizeRequest

XtGeometryResult XtMakeResizeRequest(*widget, width, height, width_return, height_return*)

```
Widget widget;
Dimension width, height;
Dimension *width_return, *height_return;
```

widget Specifies the widget.

width Specifies the desired width.

height Specifies the desired height.

width_return Returns the allowed width.

height_return Returns the allowed height.

XtMakeResizeRequest makes a resize request from a widget. A child widget can use this routine to request a geometry change to the parent widget. This routine is an interface to **XtMakeGeometryRequest**. It creates an **XtWidgetGeometry** structure and specifies that width and height should change. At this time, the geometry manager can modify any of the other window attributes, such as position or stacking order, to satisfy the resize request.

If **XtGeometryAlmost** is returned, the return value contains a compromise width and height. If this compromise is acceptable, the widget should make an immediate **XtMakeResizeRequest** for the compromised value for width and height.

If the widget is not interested in **XtGeometryAlmost**, it passes **NULL** for *width_return* and *height_return*.

¦ Copyright IBM Corp. 1985, 1991

4.17.115 XtMalloc

char *XtMalloc(*size*)
Cardinal *size*;

size Specifies the number of bytes.

XtMalloc allocates storage. It returns a pointer to a block of storage which is at least the specified size bytes. If there is insufficient memory to allocate the new block, **XtMalloc** calls **XtErrorMsg**.

¦ Copyright IBM Corp. 1985, 1991

4.17.116 XtManageChild

void XtManageChild(*child*)
Widget *child*;

child Specifies the child widget.

XtManageChild adds a single child to a parent widget list of managed children. It is called after **XtCreateWidget**.

XtManageChild constructs a **WidgetList**; then, it calls **XtManageChildren**.

¦ Copyright IBM Corp. 1985, 1991

4.17.117 XtManageChildren

typedef Widget *WidgetList;

void XtManageChildren(*children, num_children*)
WidgetList *children*;
Cardinal *num_children*;

children Specifies a list of child widgets.

num_children Specifies the number of children.

XtManageChildren adds a list of widgets to the geometry-managed, displayable, subset of its composite parent widget. This routine is called after the child widgets have been created with **XtCreateWidget**. **XtManageChildren** performs the following:

Issues an error if all the children do not have the same paren

Returns immediately if the common parent is being destroyed

Otherwise, for each unique child on the list, it:

compact.

Ignores the child if the child is already managed or is bein destroyed

Marks the child as managed and increments the *num_mapped_children* field of the parent if the child's *map_when_managed* value is **True**.

If the parent is realized after all children have been marked, **XtManageChildren** makes some of the newly managed children visible in the following manner:

Calling the *change_managed* routine of the parent widget

Calling **XtRealizeWidget** for each previously unmanaged, unrealized child

Mapping each previously unmanaged child with a *map_when_managed* value of **True**.

¦ Copyright IBM Corp. 1985, 1991

4.17.118 XtMapWidget

XtMapWidget(*widget*)
    Widget *widget;*

widget Specifies the widget.

XtMapWidget maps a widget explicitly.

¦ Copyright IBM Corp. 1985, 1991

4.17.119 XtMergeArgLists

ArgList XtMergeArgLists(*args1, num_args1, args2, num_args2*)
ArgList *args1*;
Cardinal *num_args1*;
ArgList *args2*;
Cardinal *num_args2*;

args1 Specifies the first argument list.

num_args1 Specifies number of arguments in the first argument list.

args2 Specifies the second argument list.

num_args2 Specifies the number of arguments in the second argument list.

XtMergeArgLists merges two **ArgList** structures. It allocates sufficient storage to hold the combined **ArgList** structures and copies them into that space. **XtMergeArgLists** does not check the **ArgList** structures for duplicate entries.

Use **XtFree** to deallocate the storage space after this routine completes.

¦ Copyright IBM Corp. 1985, 1991

4.17.120 XtMoveWidget

void XtMoveWidget(*widget, x, y*)
Widget *widget*;
Position *x*;
Position *y*;

widget Specifies the widget.

x, y Specifies the new *x,* and *y* coordinates.

XtMoveWidget moves a sibling widget of the child making the *geometry request*. It writes the new *x* and *y* values into the widget. If the widget is realized, **XtMoveWidget** issues an **XMoveWindow** call on the window of the widget.

¦ Copyright IBM Corp. 1985, 1991

4.17.121 XtNameToWidget

WidgetXtNameToWidget(*reference, names*);
Widget *reference*;
String *names*;

reference Specifies the widget from which to start the search.

names Specifies the fully qualified name of the desired widget.

XtNameToWidget translates a widget name to a widget instance. It searches for the following:

A widget whose name is the first component in the *names* argument. (This argument may contain more than one widget name for widgets that are not direct children of the specified reference widget.)

And, a pop-up child of the specified reference widget

Or, a normal child of the specified reference, if the reference widge is a **compositeWidgetClass** widget.

Then, it uses theat widget as the new reference widget and repeats the search after deleting the first component from the *names* argument.

If **XtNameToWidget** cannot find the specified widget, it returns **NULL**.

If more than one child of the reference widget matches the name, **XtNameToWidget** returns any of the children.

The Intrinsics does not require unique names for the children of a widget. If the *names* argument contains more than one component and more than one child matches the first component, **XtNameToWidget** returns **NULL** if the single branch that it follows does not contain the named widget.

XtNameToWidget does not back up and follow other matching branches of the widget tree.

¦ Copyright IBM Corp. 1985, 1991

4.17.122 XtNew

type ***XtNew**(*type*)
type;

type Specifies a previously declared data type.

XtNew allocates storage for a new instance of a data type. It returns a pointer to the allocated storage. If there is insufficient memory to allocate the new block, it calls **XtErrorMsg**.

XtNew calls **XtMalloc** with the following arguments:

((type *) XtMalloc((unsigned) sizeof(type))

¦ Copyright IBM Corp. 1985, 1991

4.17.123 XtNewString

String XtNewString(*string*)
String *string*

string Specifies a previously declared string.

XtNewString copies an instance of a string. It returns a pointer to the allocated storage. If there is insufficient memory to allocate the new block, this routine calls **XtErrorMsg**.

XtNewString calls **XtMalloc** with the following arguments:

```
(strcpy(XtMalloc((unsigned)strlen(str) + 1), str))
```

¦ Copyright IBM Corp. 1985, 1991

4.17.124 XtNextEvent

void XtNextEvent(*event_return*)
XEvent **event_event*;

event_return Returns the event information to the specified event structure.

XtNextEvent returns the value from the header of the input queue.

If there is no input on the X-Windows input queue, **XtNextEvent** flushes the output buffer. It waits for an event while looking at the other input sources and timeout values. **XtNextEvent** calls the callback procedures triggered by the timeout values.

XtInitialize must be called before using **XtNextEvent**.

Note: This routine has been replaced by **XtAppNextEvent**.

¦ Copyright IBM Corp. 1985, 1991

4.17.125 XtNumber

Cardinal XtNumber(*array*)
      ArrayVariable *array*;

array Specifies a fixed-size array.

XtNumber determines the number of elements in a fixed-size array. It returns the number of elements in the specified argument lists, resources lists, and other counted arrays.

¦ Copyright IBM Corp. 1985, 1991

4.17.126 XtOffset

Cardinal XtOffset(*pointer_type, field_name*)
Type *pointer_type*;
Field *field_name*;

pointer_type Specifies a type declared as a pointer to the structure.

field_name Specifies the name of the field for which to calculate the byte offset.

XtOffset determines the byte offset of a resource field within a structure. It calculates (the offset) from the beginning of a widget. WHAT can be used at compile time in static initializations.

¦ Copyright IBM Corp. 1985, 1991

4.17.127 XtOpenDisplay

Display *XtOpenDisplay(*application_context, display_string, application_name,
application_class, options, num_options, arg*

XtAppContext *application_context*;
String *display_string*;
String *application_name*;
String*application_class*;
XrmOptionDescRec **options*;
Cardinal *num_options*;
Cardinal **argc*;
String **argv*;

application_context	Specifies the application context.
display_string	Specifies the display string. Each display can be in only one application context.
application_name	Specifies the name of the application instance.
application_class	Specifies the class name of this application, usually the generic name for all instances of this application.
options	Specifies how to parse the command line for any application-specific resources. The *options* is in **XrmParseCommand**. For further information, see Chapter 1, "Using X-Windows" in topic 1.0.
num_options	Specifies the number of entries in the options list.
argc	Specifies a pointer to the number of command line parameters.
argv	Specifies the command line parameters.

XtOpenDisplay opens, initializes, and adds a display to an application context. It calls **XOpenDisplay** with the display name.

If *display_string* is **NULL**, **XtOpenDisplay** uses the current value of the **-display** specified in *argv*.

If display name is not specified in *argv*, the **DISPLAY** environment or the user's default display is used.

After a display name is found, **XtOpenDisplay** calls **XtDisplayInitialize** specifying the display name and application name.

The application name is the **-name** option, if specified in *argv*.

If an application name is not specified, **XtOpenDisplay** uses the *application_name* parameter passed to it.

If the application name is **NULL**, it passes the last component of *argv*[0].

Values for the **-display** and **-name** options found in the *argv* parameter to **XtOpenDisplay** override the values for the same parameter in the **XtDisplayInitialize** routine.

¦ Copyright IBM Corp. 1985, 1991

If **XtOpenDisplay** is successful, it returns the opened display. If it is not successful, it returns **NULL**

¦ Copyright IBM Corp. 1985, 1991

4.17.128 XtOrderProc

```
typedef Cardinal(*XtOrderProc)(Widget);
      Widget widget;
```

widget Specifies the widget.

XtOrderProc is the *insert_procedure* type for a composite widget instance. This routine is useful when composite widgets need a specific order for their children widgets.

¦ Copyright IBM Corp. 1985, 1991

4.17.129 XtOverrideTranslations

void XtOverrideTranslations(*widget, translations*)
Widget *widget*;
XtTranslations *translations*;

widget Specifies the widget for merging the new translations.

translations Specifies the compiled translation table to merge.

XtOverrideTranslations overwrites existing translations with new translations. The old translations do not exist once they have been written over. If the new translations contain an event or event sequence that already exists for the widget, the new translation overrides the existing widget.

¦ Copyright IBM Corp. 1985, 1991

4.17.130 XtOwnSelection

Boolean XtOwnSelection (*widget, selection, time, convert_proc, lose_selection,*
Widget *widget*;
Atom *selection*;
Time *time*;
XtConvertSelectionProc *convert_proc*;
XtLoseSelectionProc *lose_selection*;
XtSelectionDoneProc *done_proc*;

widget Specifies the widget that will become the owner.

selection Specifies an atom that describes the type of the selection (for example, **XA_PRIMARY**, **XA_SECONDARY**, or **XA_CLIPBOARD**).

time Specifies the timestamp. This should be the timestamp of the event that triggered ownership. (**CurrentTime** cannot be used.)

convert_proc Specifies the procedure to call when the current value of the selection is requested.

lose_selection Specifies the procedure to call when the widget loses selection ownership. Specifies NULL if the owner is not interested in being called back.

done_proc Specifies the procedure to call after the requestor has received the selection. Specifies NULL if the owner is not interested in being called back.

XtOwnSelection sets the selection owner when using atomic transfers. It informs the Intrinsics selection mechanism that a widget believes it owns a selection.

XtOwnSelection returns

True if the widget becomes the selection owner.

False if the widget does not become the selection owner.

This widget may fail to become the selection owner for one of the following reasons:

If another widget asserts ownership at a time later the specifie timestamp. (The specified *timestamp* triggers this event.)

If this widget surrenders ownership of the selection voluntarily

The lose_selection procedure is not called if the widget does not obtain selection ownership.

¦ Copyright IBM Corp. 1985, 1991

4.17.131 XtParent

Widget XtParent (*widget*)
Widget *widget*

widget Specifies the widget.

XtParent returns the parent widget for the specified widget.

¦ Copyright IBM Corp. 1985, 1991

4.17.132 XtParseAcceleratorTable

XtAccelerators XtParseAcceleratorTable(*source*)
String *source*;

source Specifies the accelerator table to compile.

XtParseAccelerTable parses an accelerator table into the opaque internal representation. An accelerator is a translation table that is bound with its actions in the context of a particular widget.

¦ Copyright IBM Corp. 1985, 1991

4.17.133 XtParseTranslationTable

XtTranslations XtParseTranslationTable(*table*)
String *table*;

table Specifies the translation table to compile.

XtParseTranslationTable compiles a translation table into the opaque internal representation of type **XtTranslations**. If an empty translation table is required for any purpose, it can be obtained by calling **XtParseTranslationTable** and passing an empty string.

¦ Copyright IBM Corp. 1985, 1991

4.17.134 XtPeekEvent

Boolean XtPeekEvent(*event_return*)
XEvent **event_return*;

event_return Returns the event information to the specified event structure.

XtPeekEvent returns the value from the front of the input queue without removing it from the queue.

If there is an event in the queue for the default application context it fills in the event and returns a non-zero value.

If there is no X input in the queue, **XtPeekEvent** flushes the output buffer and blocks until input is available, possibly calling some timeout callbacks in the process.

If the input is an event, it fills in the event and returns a non-zer value.

If the input is for an alternate input source, it returns zero

XtInitialize must be called before using this routine.

Note: This routine has been replaced by **XtAppPeekEvent**.

¦ Copyright IBM Corp. 1985, 1991

4.17.135 XtPending

Boolean XtPending();

XtPending determines if the input queue has events pending. It returns a nonzero value if X events or other input sources in the default application context are pending. If there are no events pending, it flushes the output buffer and returns a zero value.

XtInitialize must be called before using this routine.

Note: This routine has been replaced by **XtAppPending**.

¦ Copyright IBM Corp. 1985, 1991

4.17.136 XtPopdown

void XtPopdown(*popup_shell*)
Widget *popup_shell*;

popup_shell Specifies the widget shell.

XtPopdown unmaps a pop-up from within an application. It does the following:

Calls **XtCheckSubclass** to ensure *popup_shell* is a subclass of **Shell**

Generates an error if *popup_shell* is currently *popped_up*

Unmaps the window of the *popup_shell*

Calls **XtRemoveGrab** if the *grab_kind* of the *popup_shell* is **XtGrabNonexclusive** or **XtGrabExclusive**

Sets *popup_shell popped_up* field to **False**

Calls the callback procedures on the *popdown_callback* list of the shell.

¦ Copyright IBM Corp. 1985, 1991

4.17.137 XtPopup

void XtPopup(*popup_shell, grab_kind*)
Widget *popup_shell*;
XtGrabKind *grab_kind*;

popup_shell Specifies the widget shell.

grab_kind Specifies how user events should be constrained.

XtPopup maps a pop-up from within an application. It does the following:

Calls **XtCheckSubclass** to ensure *popup_shell* is a subclass of **Shell**

Generates an error if the *popped_up* field of the shell is already **True**

Calls the callback procedures on the *popup_callback* on the list of the shell

Sets the shell *popped_up* field to **True** and the shell *spring_loaded* field to **False**

Calls the *create_popup_child* field of the shell with *popup_shell* as the parameter, if the *create_popup_child* field of the shell is non-NULL

Calls **XtAddGrab**(*popup_shell*, (*grab_kind* == **XtGrabExclusive**),**False**), if *grab_kind* is **XtGrabNonexclusive** or **XtGrabExclusive**

Calls **XtRealizeWidget** with the *popup_shell* specified

Calls **XMapWindow** with the *popup_shell*.

¦ Copyright IBM Corp. 1985, 1991

4.17.138 XtProc

typedef void (*XtProc)();

XtProc is the class initialization procedure type used by the Toolkit. A widget class can declare that this procedure be be called once automatically by the Toolkit.

Specify **NULL** for *class_initialize* to indicate that a widget class does not have a class initialization procedure.

¦ Copyright IBM Corp. 1985, 1991

4.17.139 XtProcessEvent

void XtProcessEvent(*mask*)
XtInputMask **mask*;

mask Specifies the type of input to process.

XtProcessEvent processes one input event, timeout, or alternate input source (depending on the value of the mask). **XtProcessEvent** waits for an event of the specified type if one is not available.

XtInitialize must be called before using **XtProcessEvent**.

Note: This routine has been replaced by **XtAppProcessEvent**.

¦ Copyright IBM Corp. 1985, 1991

4.17.140 XtQueryGeometry

XtGeometryResult XtQueryGeometry (*widget, intended, preferred_return*)
Widget *widget*;
XtWidgetGeometry **intended, *preferred_return*;

widget	Specifies the widget.
intended	Specifies changes the parent intends for the geometry of the child geometry.
preferred_return	Returns the preferred geometry of the child widget.

XtQueryGeometry queries the preferred geometry of a child widget. This routine is called by the parent widget to set any changes in the geometry of the child widget. **XtQueryGeometry** is called after the following takes place:

The parent widget sets the (intended) changes in the correspondin fields

Then, the parent widget sets the corresponding bits i *intended.request_mode*

XtQueryGeometry clears all the bits in the *preferred_return->request_mode* and checks the *query_geometry* field of the specified widget class record.

If *query_geometry* is not **NULL**, **XtQueryGeometry** calls the *query_geometry* procedure passing the specified widget, *intended* and *preferred_return* structures as arguments.

If the *intended argument* is **NULL**, this routine is replaced with a pointer to an **XtWidgetGeometry** structure with *request_mode* set to zero before calling *query_geometry*.

The *query_geometry* procedure type is **XtGeometryHandler**.

After calling the *query_geometry* procedure or if the *query_geometry* is **NULL**, **XtQueryGeometry** examines all the unset bits in *preferred_return->request_mode* and sets the corresponding fields in *preferred_return* to the current values from the widget instance. If **CWStackMode** is not set, the *stack_mode* is **XtSMDontChange**. Then, **XtQueryGeometry** returns one of the following values from the *query_geometry* procedure:

XtGeometryYes which indicates that the proposed geometry is acceptable without modification.

XtGeometryAlmost which indicates that both the parent and the child required at least one common field and the child requirement does not match the parent requirement or the child required a field that the parent may also require. (For example, when a child defines a new field, a parent must redefine its own.)

XtGeometryNo which indicates that the parent and the child required a field and the child suggested its current value as the preferred value.

Whether the caller chooses to ignore the return value or the reply mask,

¦ Copyright IBM Corp. 1985, 1991

the reply structure should contain the complete geometry information for the child.

Parent widgets should call **XtQueryGeometry** in their layout routine, and, in any routine, after the *change_managed* procedure is used.

¦ Copyright IBM Corp. 1985, 1991

4.17.141 XtRealizeProc

```
typedef void (*XtRealizeProc)(Widget, XtValueMask*, XSetWindowAttributes*);
      Widget widget;
      XtValueMask *value_mask;
      XSetWindowAttributes *attributes;
```

widget Specifies the widget.

value_mask Specifies the fields from the attributes structure to use.

attributes Specifies the window attributes to use in the **XCreateWindow** call.

XtRealizeProc is the realize procedure type for a widget class. This procedure must create a widget's window.

¦ Copyright IBM Corp. 1985, 1991

4.17.142 XtRealizeWidget

void XtRealizeWidget(*widget*)
Widget *widget;*

widget Specifies the widget.

XtRealizeWidget realizes a widget instance. If the widget is already realized, **XtRealizeWidget** returns. If the widget is not realized, the routine performs the following:

1. Binds all action names in the widget's translation table to procedures. (See "Using Action Tables" in topic 4.16.1.)

2. Makes a post-order traversal of the widget tree that is rooted at the specified widget, then calls the *change_managed* procedure of each composite widget that has one or more managed children.

3. Constructs an **XSetWindowAttributes** structure filled in with information derived from the Core widget fields, then calls the realize procedure for the widget, which adds any widget-specific attributes and creates the X-window. While filling in the mask and corresponding **XSetWindowAttributes** structure, the **XtRealizeWidget** routine sets the following fields based on information in the Core widget fields:

   The *background_pixmap* (or *background_pixel* if *background_pixmap* is **NULL**) is filled in from the corresponding field in the **Core** structure.

   The *border_pixmap* (or *border_pixel* if *border_pixmap* is **NULL**) is filled in from the corresponding field in the **Core** structure.

   The *event_mask* is filled in based on the event handlers registered, the event translations specified, whether expose is not set to **NULL**, and whether visible_interest is set to **True**.

   The *bit_gravity* is set to **NorthWestGravity** if the expose field is **NULL**.

   The *do_not_propagate* mask is set to propagate all pointer and keyboard events anywhere inside it, including on top of children widgets, as long as children do not specify a translation for the event.

   All other fields in attributes and their corresponding bits in *value_mask* can be set by the realize procedure.

4. If the widget is not a subclass of **compositeWidgetClass**, **XtRealizeWidget** returns. Otherwise, it continues and does the following:

   a. Descends recursively to each of the widget's managed children and calls the realize procedures. Primitive widgets that instantiate children are responsible for realizing those children themselves.

   b. Maps all of the managed children windows that have the **mapped_when_managed** field set to **True**. If a widget is managed but *mapped_when_managed* is **False**, the widget is allocated visual space, but is not displayed.

¦ Copyright IBM Corp. 1985, 1991

If the widget is the top-level widget, which by definition does not have a parent, and *map_when_managed* is true, **XtRealizeWidget** maps the widget window.

¦ Copyright IBM Corp. 1985, 1991

4.17.143 XtRealloc

char *XtRealloc(*ptr, num*)
char **ptr*;
Cardinal *num*;

ptr Specifies the pointer to old storage.

num Specifies the number of bytes in new storage required.

XtRealloc changes the size of an allocated block of storage, sometimes moving it. Then, it copies the old contents (or as much as will fit) into the new block and frees the old block.

If there is insufficient memory to allocate the new block, **XtRealloc** calls **XtErrorMsg**.

If *ptr* is **NULL**, **XtRealloc** calls **XtMalloc**, which allocates the new storage without copying the old contents.

¦ Copyright IBM Corp. 1985, 1991

4.17.144 XtRegisterCaseConverter

```
void XtRegisterCaseConverter(*display, proc, start, stop);
      Display *display;
      XtCaseProc proc;
      KeySym start;
      KeySym stop;
```

display Specifies the display providing the key events.

proc Specifies the **XtCaseProc** for the conversions.

start Specifies the first KeySym for which this converter is valid.

stop Specifies the last KeySym for which this converter is valid.

XtRegisterCaseConverter registers a specified case converter. The *start* and *stop* arguments provide the inclusive range of KeySyms for which this converter will be called. The new converter overrides previous converters for KeySyms in the specified range.

There is no interface to remove converters. You need to register an identity converter. When a new converter is registered, the Intrinsics refreshes the keyboard state, if necessary. The default converter understands case conversion for all KeySyms defined in the core protocol.

¦ Copyright IBM Corp. 1985, 1991

4.17.145 XtReleaseGC

void XtReleaseGC(*widget, gc*);
Widget *widget*;
GC *gc*;

widget Specifies the widget.

gc Specifies the GC to be deallocated.

XtReleaseGC deallocates a shared GC when it is no longer needed. References to sharable GCs are counted and a free request is generated to the server when the last user of a specified GC destroys the GC.

¦ Copyright IBM Corp. 1985, 1991

4.17.146 XtRemoveAllCallbacks

void XtRemoveAllCallbacks(*widget, callback_name,*)
Widget *widget*;
String *callback_name*;

widget Specifies the widget.

callback_name Specifies the callback list to be removed.

XtRemoveAllCallbacks deletes all callback procedures from a specified widget callback list. It also frees the storage associated with the callback list.

¦ Copyright IBM Corp. 1985, 1991

4.17.147 XtRemoveCallback

void XtRemoveCallback(*widget, callback_name, callback, client_data*)
Widget *widget*;
String *callback_name*;
XtCallbackProc *callback*;
caddr_t *client_data*;

widget Specifies the widget.

callback_name Specifies the callback list from which the callback procedure will be removed.

callback Specifies the callback procedure.

client_data Specifies the client data to match on the registered callback procedure.

XtRemoveCallback deletes a callback procedure from a specified widget callback list only if both the procedure and the client data match.

¦ Copyright IBM Corp. 1985, 1991

4.17.148 XtRemoveCallbacks

void XtRemoveCallbacks(*widget, callback_name, callbacks*)
Widget *widget*;
String *callback_name*;
XtCallbackList *callbacks*;

widget Specifies the widget.

callback_name Specifies the callback list from which the callback procedures will be removed.

callbacks Specifies the null-terminated list of callback procedures and corresponding client data.

XtRemoveCallbacks deletes a list of callback procedures from a specified widget callback list.

¦ Copyright IBM Corp. 1985, 1991

4.17.149 XtRemoveEventHandler

void XtRemoveEventHandler(*widget, event_mask, nonmaskable, proc, client_data*)
Widget *widget*;
XtEventMask *event_mask*;
Boolean *nonmaskable*;
XtEventHandler *proc*;
caddr_t *client_data*;

widget Specifies the widget to register for this procedure.

event_mask Specifies the event mask to unregister for this procedure.

nonmaskable Specifies a Boolean value that indicates whether this procedure should be removed on the nonmaskable events.

proc Specifies the event handler procedure to be removed.

client_data Specifies the client data registered for this procedure.

XtRemoveEventHandler removes a previously registered event handler. It stops the specified procedure from receiving the specified events. The request is ignored if client data does not match the value given in the call to **XtAddEventHandler**. If the widget is realized, **XtRemoveEventHandler** calls **XSelectInput**, if necessary.

The nonmaskable events include **GraphicsExpose, NoExpose, SelectionClear, SelectionRequest, SelectionNotify, ClientMessage**, and **MappingNotify**.

To stop a procedure from receiving any events that entirely remove it from the widget *event_table*, call **XtRemoveEventHandler** with an *event_mask* of **XtAllEvents** and with nonmaskable **True**.

¦ Copyright IBM Corp. 1985, 1991

4.17.150 XtRemoveGrab

void XtRemoveGrab(*widget*)
Widget *widget*;

widget Specifies the widget to remove from the modal cascade.

XtRemoveGrab removes the redirection of user input to a modal widget. It removes widgets from the modal cascade starting at the most recent widget and including the specified widget. If the specified widget is not on the modal cascade, it issues an error message.

¦ Copyright IBM Corp. 1985, 1991

4.17.151 XtRemoveInput

void XtRemoveInput(*id*)
XtInputId ***id**;

id Specifies the ID returned from the corresponding **XtAppAddInput** call.

XtRemoveInput discontinues a source of input by causing the Intrinsics read routine to stop watching for input from the input source.

¦ Copyright IBM Corp. 1985, 1991

4.17.152 XtRemoveRawEventHandler

void XtRemoveRawEventHandler(*widget, event_mask, nonmaskable, proc, client_dat*
Widget *widget*;
EventMask *event_mask*;
Boolean *nonmaskable*;
XtEventHandler *proc*;
caddr_t *client_data*;

widget Specifies the widget to register for this procedure.

event_mask Specifies the event mask to unregister for this procedure.

nonmaskable Specifies a Boolean value that indicates whether this procedure should be removed on the nonmaskable events.

proc Specifies the event handler procedure to be registered.

client_data Specifies the client data registered.

XtRemoveRawEventHandler removes a previously registered raw event handler. It stops the specified procedure from receiving the specified events. This routine does not affect the widget masks because it is a raw event handler. And, it never causes a call to **XSelectInput**.

The nonmaskable events include **GraphicsExpose, NoExpose, SelectionClear, SelectionRequest, SelectionNotify, ClientMessage**, and **MappingNotify**.

¦ Copyright IBM Corp. 1985, 1991

4.17.153 XtRemoveTimeOut

void XtRemoveTimeOut(*timer*)
XtIntervalId *timer*;

timer Specifies the ID for the timeout request to be destroyed.

XtRemoveTimeout clears a timeout value by removing the timeout. (Timeouts are removed automatically once they are triggered.)

¦ Copyright IBM Corp. 1985, 1991

4.17.154 XtRemoveWorkProc

void XtRemoveWorkProc(*id*)
XtWorkProcId *id*;

id Specifies the work procedure to remove.

XtRemoveWorkProc removes the specified background work procedure.

¦ Copyright IBM Corp. 1985, 1991

4.17.155 XtResizeWidget

void XtResizeWidget(*widget, width, height, border_width*)
Widget *widget*;
Dimension *width*;
Dimension *height*;
Dimension *border_width*;

widget Specifies the widget.

width Specifies the width of the new widget.

height Specifies the width of the new widget.

border_width Specifies the size of the new widget.

XtResizeWidget resizes a sibling widget of the child making the geometry request. It returns immediately if the new *width, height,* and *border_width* values are the same as the old values. Otherwise, it writes the new values into the widget.

If the widget is realized, **XtResizeWidget** issues an **XConfigureWindow** call on the window of the widget.

If the new *width* or *height* are different from the old values, **XtResizeWidget** calls the resize procedure of the widget to notify it of the size change.

¦ Copyright IBM Corp. 1985, 1991

4.17.156 XtResizeWindow

void XtResizeWindow(*widget*)
Widget *widget*

widget Specifies the widget.

XtResizeWindow resize a child widget that already has the new values for its width, height, and border width. This routine calls **Xlib XConfigureWindow** to make the window of the specified widget match the width, height, and border width of the specified values. The configure request is done unconditionally because there is no check on the values. The widget's resize procedure is not called. Use **XtResizeWidget** under most normal conditions.

¦ Copyright IBM Corp. 1985, 1991

4.17.157 XtScreen

Screen *XtScreen
Widget *widget*

XtScreen returns the screen pointer for the specified widget.

¦ Copyright IBM Corp. 1985, 1991

4.17.158 XtSelectionCallbackProc

```
typedef void (*XtSelectionCallbackProc)(Widget, caddr_t, Atom*, Atom*,
                    caddr_t, unsigned long*, int*);
      Widget widget;
      caddr_t client_data;
      Atom *selection;
      Atom *type;
      caddr_t value;
      unsigned long *length;
      int *format;
```

widget Specifies the widget that requested the selection value.

client_data Specifies a value passed in by the widget when the selection was requested.

selection Specifies the type of selection that was requested.

type Specifies the representation type of the selection value (for example, **XA_STRING**).

value Specifies a pointer to the selection value.

length Specifies the number of elements in value.

format Specifies the size in bits of the data elements of value.

XtSelectionCallbackProc is called by the Intrinsics selection mechanism to deliver the requested selection to the requestor.

The *type* refers to the type used to represent the target.

XT_CONVERT_FAIL indicates that the selection conversion failed because the selection owner did not respond within the selection timeout interval provided by the Intrinsics.

The requesting client owns the storage allocated for this routine. Use **XtFree** to de-allocate the storage space when this routine completes.

¦ Copyright IBM Corp. 1985, 1991

4.17.159 XtSelectionDoneProc

```
typedef void(*XtSelectionDoneProc)(Widget, Atom*);
      Widget widget;
      Atom *selection;
      Atom *target;
```

widget Specifies the widget that owns the converted selection.

selection Specifies the atom that describes the selection type that was converted.

target Specifies the target type to which the conversion was done.

XtSelectionDoneProc is called by the Intrinsics selection mechanism to inform the selection owner when a selection requestor has retrieved a selection value successfully.

Once the selection owner registers an **XtSelectionDoneProc**, the procedure will be called once for each conversion that it performs. This procedure is called after the converted value has been transferred successfully to the requestor.

The selection owner that registers an **XtSelectionDoneProc**, also owns the storage containing the converted selection value.

¦ Copyright IBM Corp. 1985, 1991

4.17.160 XtSetArg

XtSetArg(*arg, name, value*)
Arg *arg*;
String *name*;
XtArgVal *value*;

arg Specifies the name-value pair to set.

name Specifies the name of the resource.

value Specifies the value of the resource if less than or equal to the size of an **XtArgVal** structure. Otherwise, *value* specifies the address of the resource.

XtSetArg sets values in an **ArgList**. This function is usually specified in a stylized manner to minimize the probability of making a mistake. For example,

```
Arg args[20];
int n;

n = 0;
XtSetArg(args[n], XtNheight, 100);     n++;
XtSetArg(args[n], XtNwidth, 200);      n++;
XtSetValues(widget, args, n);
```

An application could also declare the argument list and use the **XtNumber** routine. For example:

```
static Args args[]={
     {XtNheight, (XtArgVal) 100},
     {XtNwidth, (XtArgVal) 200},
};
XtSetValues(Widget, args, XtNumber(args));
```

Note: Do not use auto-increment or auto-decrement in the first argument to **XtSetArg**. As currently implemented, this macro de-references the first argument twice.

¦ Copyright IBM Corp. 1985, 1991

4.17.161 XtSetErrorHandler

void XtSetErrorHandler(*handler*)
XtErrorHandler *handler*;

handler Specifies the new fatal error procedure.

XtSetErrorHandler registers a procedure to call under fatal error conditions. The default error handler provided by the Intrinsics is **_XtError**. It prints the message to standard error and terminates the application.

Fatal error message handlers should not return. (If a fatal error message handler returns, subsequent Toolkit behavior is undefined.)

¦ Copyright IBM Corp. 1985, 1991

4.17.162 XtSetErrorMsgHandler

void XtSetErrorMsgHandler(*msg_handler*)
XtErrorMsgHandler *msg_handler;*

msg_handler Specifies the new fatal error procedure.

XtSetErrorMsgHandler registers a procedure to call under fatal error conditions. The default error handler provided by the Intrinsics constructs a string from the error resource database and calls **_XtError**.

Fatal error message handlers should not return. (If a fatal error message handler returns, subsequent Toolkit behavior is undefined.)

¦ Copyright IBM Corp. 1985, 1991

4.17.163 XtSetKeyboardFocus

XtSetKeyboardFocus(*subtree, descendant*)
Widget *subtree, descendant*;

subtree Specifies the subtree of the hierarchy for which the keyboard focus is to be set.

descendant Specifies the widget in the subtree structure which will receive the keyboard event or it specifies **None**.

XtSetKeyboardFocus redirects keyboard input to a child of a composite widget without calling **XSetInputFocus**.

If a future **KeyPress** or **KeyRelease** event occurs on the specified widget, **XtSetKeyboardFocus** causes **XtDispatchEvent** to remap and send the event to the specified descendant widget.

¦ Copyright IBM Corp. 1985, 1991

4.17.164 XtSetKeyTranslator

void XtSetKeyTranslator(*display, proc*);
Display **display*;
XtKeyProc **proc*;

display Specifies the display from which to translate the events.

proc Specifies the procedure that is to perform key translations.

XtSetKeyTranslator registers a key translator. Its sets the specified procedure as the current key translator. The default translator is **XtTranslateKey**, an **XtKeyProc** that uses Shift and Lock modifiers with the interpretations defined by the core protocol.

XtSetKeyTranslator is provided for new translators to obtain default KeyCode-to-KeySym translations and for reinstalling the default translator.

¦ Copyright IBM Corp. 1985, 1991

4.17.165 XtSetMappedWhenManaged

void XtSetMappedWhenManaged(*widget, map_when_managed*)
Widget *widget*;
Boolean *map_when_managed*;

widget Specifies the widget.

map_when_managed Specifies a Boolean value.

XtSetMappedWhenManaged changes the widget *map_when_managed* field.

If the widget is realized and managed and the new value of *map_when_managed* is **True**, this routine maps the window. If the widget is realized and managed and the new value of *map_when_managed* is **False**, this routine unmaps the window.

As an alternative to **XtSetMappedWhenManaged** to control mapping, a client can set *mapped_when_managed* to **False**, and call **XtMapWidget** and **XtUnmapWidget** explicitly.

¦ Copyright IBM Corp. 1985, 1991

4.17.166 XtSetSelectionTimeout

void XtSetSelectionTimeout(*timeout*)
     unsigned long *timeout*

XtSetSelectionTimeout set the Instrinsics selection timeout. The selection timeout is the time during which two applications (that are communicating) must respond to one another. If the applications do not respond within this interval, the Intrinsics aborts the selection request. The default value of the selection timeout is 5 second.

¦ Copyright IBM Corp. 1985, 1991

4.17.167 XtSetSensitive

void XtSetSensitive(*widget, sensitive*)
Widget *widget*;
Boolean *sensitive*;

widget Specifies the widget.

sensitive Specifies a Boolean value that indicates whether the widget should receive keyboard and pointer events.

XtSetSensitive sets the sensitivity state of a widget. First, it calls the **XtSetValues** on the current widget with an argument list specifying that the sensitive field should change to the new value. Then, it recursively propagates the new value down the managed children tree by calling **XtSetValues** on each child to set the *ancestor_sensitive* to the new value if the new values for sensitive and the child values for *ancestor_sensitive* are not the same.

XtSetSensitive calls **XtSetValues** to change sensitive and ancestor sensitive. With these changes, the widget set_values procedure takes the display actions necessary, for example, greying out or stippling the widget. **XtSetSensitive** maintains that if the parent widget has sentive or ancestor_sensitive **False**, then all the children widgets also have the ancestor_sensitive set to **False**.

¦ Copyright IBM Corp. 1985, 1991

4.17.168 XtSetSubvalues

void XtSetSubvalues (*base, resources, num_resources, args, num_args*)
caddr_t *base*;
XtResourceList *resources*;
Cardinal *num_resources*;
ArgList *args*;
Cardinal *num_args*;

base Specifies the base address of the subpart data structure where the resources should be written.

resources Specifies the current non-widget resource values.

num_resources Specifies the number of resources in the resource list.

args Specifies a variable length argument list of name and value pairs that contains the resources to be modified and their new values. The resources and values passed are dependent on the subpart of the widget being modified.

num_args Specifies the number of resources in argument list.

XtSetSubvalues sets the current value of a non-widget resource associated with a widget instance.

¦ Copyright IBM Corp. 1985, 1991

4.17.169 XtSetValues

void XtSetValues(*widget, args, num_args*)
Widget *widget*;
ArgList *args*;
Cardinal *num_args*;

widget Specifies the widget.

args Specifies a variable length argument list of name and value pairs that contain the resources to be modified and their new values. The resources and values passed are dependent on the widget being modified.

num_args Specifies the number of resources in the argument list.

XtSetValues modifies the current value of a resource associated with a widget instance. It starts with the resources specified for the Core widget fields and proceeds down the subclass chain to the widget. At each stage, it writes the new value if specified by one of the arguments. It writes the existing value to a new widget data record if no new value is specified.

XtSetValues then calls the *set_values* procedures for the widget in superclass-to-subclass order. If the widget has any non-NULL *set_values_hook* fields, these are called with the arguments immediately after the corresponding *set_values* procedure. This procedure permits subclasses to set non-widget data for **XtSetValues**.

If the parent of the widget is a subclass of **constraintWidgetClass**, **XtSetValues** also updates of the widget constraints. It starts with the constraint resources specified for **constraintWidgetClass** and proceeds down the subclass chain to the class of the parent. At each stage, **XtSetValues** writes the new value or the existing value to a new constraint record. Then, it calls the constraint *set_values* procedures from **constraintWidgetClass** down to the class of the parent. The constraint *set_values* procedures are called with widget arguments so that the widgets can adjust the desired values based on complete information about the widget.

XtSetValues determines if a geometry request is needed by comparing the current widget to the new widget. If any geometry changes are required, **XtSetValues** makes the request and the geometry manager returns one of the following:

XtGeometryYes if it resizes the widget.

XtGeometryNo if it resets the geometry fields to their original values.

XtGeometryAlmost if it calls the *set_values_almost* procedure and writes new values for the geometry fields into the new widget.

Then, **XtSetValues** repeats the process, deciding once more whether the geometry manager should be called.

Finally, if any of the *set_values* procedures returned **True**, **XtSetValues** calls **XClearArea** on the widget window.

¦ Copyright IBM Corp. 1985, 1991

4.17.170 XtSetValuesFunc

```
typedef Boolean (*XtSetValuesFunc)(Widget, Widget, Widget);
      Widget current;
      Widget request;
      Widget new;
```

current Specifies the existing widget.

request Specifies a copy of the widget asked for by the **XtSetValues** call before any class *set_values* procedures have been called.

new Specifies a copy of the widget with the new values that are allowed.

XtSetValuesFunc is the *set_values* procedure type for a widget class. This procedure should recompute resources that are changed. For example, many **GCs** depend upon foreground and background. If no recomputation is necessary and if none of the resources specific to a subclass require the window to be redisplayed when their values are changed, then you can specify **NULL** for the *set_values* field in the class record.

Like the initialize procedure, *set_values* mostly deals with the fields defined in the subclass, but it has to resolve conflicts with its superclass, especially conflicts over width and height. In this case, though, the reference widget is *request*, not *new*.

New starts with the values of *request* but has been modified by any superclass *set_values* procedures. A widget need not refer to *request* unless it must resolve conflicts between *current* and *new*. Any changes that the widget needs to make should be made in *new*. **XtSetValues** copies the *new* values back into the *current* widget instance record after all class *set_values* procedures are called.

The *set_values* procedure must return a Boolean that indicates whether the widget needs to be redisplayed. Note that a change in the geometry fields alone does not require the *set_values* procedure to return **True**; the X Server will eventually generate an **Expose** event, if necessary. After calling all the *set_values* procedures, **XtSetValues** will force a redisplay by calling **XClearArea** if any of the *set_values* procedures returned **True**. Therefore, a *set_values* procedure should not do its own redisplaying.

It is permissible to call **XtSetValues** before a widget is realized. Therefore, the *set_values* procedure must not assume that the widget is realized.

The constraint *set_values* procedure type is also **XtSetValuesFunc**. The values passed to the constraint *set_values* procedure of the parent are the same as those passed to the class of the child *set_values* procedure. A class can specify **NULL** for the *set_values* field of the **ConstraintPart** if it does not need to compute anything.

The constraint *set_values* procedure should recompute any constraint fields derived from constraint resources that are changed. Further, it should modify the widget fields as appropriate. For example, if a constraint for the maximum height of a widget is changed to a value smaller than the current height of the widget, the constraint *set_values* procedure should reset the height field in the widget.

¦ Copyright IBM Corp. 1985, 1991

4.17.171 XtSetWarningHandler

void XtSetWarningHandler(*handler*)
XtErrorHandler *handler*;

handler Specifies the new non-fatal error procedure. Warning handlers usually return.

XtSetWarningHandler routine registers a procedure to be called on non-fatal error conditions.

The default warning handler provided by X-Windows Toolkit is **_XtWarning**. It prints the message to standard error and returns to the caller.

¦ Copyright IBM Corp. 1985, 1991

4.17.172 XtSetWarningMsgHandler

void XtSetWarningMsgHandler(*msg_handler*)
XtErrorMsgHandler *msg_handler*;

msg_handler Specifies the new non-fatal error procedure. Message warning handlers usually return.

The default warning handler provided by the Intrinsics constructs a string from the error resource database and calls **XtWarning**.

¦ Copyright IBM Corp. 1985, 1991

4.17.173 XtStringProc

```
typedef void(*XtStringProc)(Widget, String);
      Widget widget;
      String *string;
```

widget Specifies the widget that the accelerators are installed on.

string Specifies the string representation of the accelerators for this widget.

XtStringProc can specify accelerators in default files, and the string representation is the same as for a translation table. However, the interpretation of the #augment and #override directives apply to what will happen when the accelerator is installed, that is, whether or not the accelerator translations will override the translations in the destination widget. The default is #augment, which means that the accelerator translations have lower priority than the destination translations. The #replace directive is ignored for accelerator tables.

¦ Copyright IBM Corp. 1985, 1991

4.17.174 XtSuperclass

WidgetClass XtSuperclass(*widget*)
     Widget *widget*;

widget Specifies the widget.

XtSuperclass obtains the superclass of a widget by returning a pointer to the superclass structure of the widget.

¦ Copyright IBM Corp. 1985, 1991

4.17.175 XtTimerCallbackProc

```
typedef void (*XtTimerCallbackProc)(caddr_t, XtIntervalID);
     caddr_t client_data;
     XtIntervalId *id;
```

client_data Specifies the client data that was registered for this procedure in **XtAppAddTimeOut**.

id Specifies the ID returned from the corresponding **XtAppAddTimeOut** call.

XtTimerCallbackProc is the type for callback procedure called when timeouts expire.

¦ Copyright IBM Corp. 1985, 1991

4.17.176 XtToolkitInitialize

void XtToolkitInitialize();

XtToolkitInitialize initializes the X Toolkit internals. If this routine is called more than once, the behavior of the toolkit is undefined.

¦ Copyright IBM Corp. 1985, 1991

4.17.177 XtTranslateCoords

void XtTranslateCoords (*widget, x, y, rootx_return, rooty_return*)
    Widget *widget*;
    Position *x, y*;
    Position **rootx_return, *rooty_return*;

widget Specifies the widget.

x, y Specifies the widget-relative coordinates *x* and *y*.

rootx_return Returns the root-relative *x* coordinate.

rooty_return Returns the root-relative *y* coordinate.

XtTranslateCoords translates an [*x, y*] coordinate pair from widget coordinates to root coordinates. This routine does not generate a server request because the required information is already in the data structure of the widget.

¦ Copyright IBM Corp. 1985, 1991

4.17.178 XtTranslateKeycode

void XtTranslateKeycode(*display, keycode, modifiers_return, keysym_return*);
Display **display*;
KeyCode *keycode*;
Modifiers *modifiers*;
Modifiers **modifiers_return*;
KeySym **keysym_return*;

display Specifies the display that the KeyCode is from.

keycode Specifies the KeyCode to translate.

modifiers Specifies the modifiers to the KeyCode.

modifiers_return Returns a mask that indicates the modifiers actually used to generate the KeySym.

keysym_return Returns the resulting KeySym.

XtTranslateKeycode registers a key translator. It invokes the currently registered KeyCode-to-KeySym translator. It passes the specified arguments directly to the currently registered KeyCode to Key Sym translator.

¦ Copyright IBM Corp. 1985, 1991

4.17.179 XtTranslations

```
typedef struct _TranslationData *XtTranslations;

typedef struct _TranslationData {
    unsigned int numEvents;
    unsigned int eventTblSize;
    EventObjPtr eventObjTbl;
    unsigned long clickTime;
    unsigned long lastEventTime;


   unsigned int numQuarks;          /* number of entries in        */
                                    quark
   unsigned int quarkTblSize;       /* total size of quark table   */
   XrmQuark* quarkTable;            /* table of quarkified         */
   StatePtr head;                   /* head of list of all         */
                                    states

} TranslationData;
```

XtTranslations is a pointer to the typedef structure **_TranslationData**.

¦ Copyright IBM Corp. 1985, 1991

4.17.180 XtUninstallTranslations

void XtUninstallTranslations (*widget*);
Widget *widget*;

widget Specifies the widget from which the translations are to be removed.

XtUninstallTranslations causes the entire translation table for widget to be removed.

¦ Copyright IBM Corp. 1985, 1991

4.17.181 XtUnmanageChild

void XtUnmanageChild(*child*)
Widget *child*;

child Specifies the child to remove.

XtUnManageChild routine removes a single child from the managed set of its parent. It constructs a widget list with a length of 1 and calls **XtUnmanageChildren**.

¦ Copyright IBM Corp. 1985, 1991

4.17.182 XtUnmanageChildren

void XtUnmanageChildren(*children, num_children*)
WidgetList *children*;
Cardinal *num_children*;

children Specifies the children to be removed.

num_children Specifies the number of the children to be removed.

XtUnManageChildren routine removes a list of children from the managed list of the parent, but does not destroy the children widgets. This routine:

Issues an error if all the children do not have the same paren

Returns immediately if the common parent is being destroyed

For each unique child on the list, **XtUnmanageChildren**:

Ignores the child or marks the child as unmanaged if the child i already unmanaged or is being destroyed.

Makes the child non-visible by unmapping it if the child is realized

Decrements the *num_mapped_children* field of the parent if the *map_when_managed* value of the widget is **True**.

Calls the *change_managed* routine of the parent of the widget after all children have been marked and the parent is realized.

¦ Copyright IBM Corp. 1985, 1991

4.17.183 XtUnmapWidget

void XtUnmapWidget (*widget*)
Widget *widget*;

widget Specifies the widget to unmap.

XtUnmapWidget unmaps a widget explicitly.

¦ Copyright IBM Corp. 1985, 1991

4.17.184 XtUnrealizeWidget

void XtUnrealizeWidget (*widget*)
Widget *widget*;

widget Specifies the widget.

XtUnrealizeWidget destroys the windows associated with a widget and its descendants (recursively down the widget tree). To recreate the windows at a later time, call **XtRealizeWidget** again.

If the widget was managed, it will be unmanaged automatically before its window is freed.

¦ Copyright IBM Corp. 1985, 1991

4.17.185 XtWarning

void XtWarning(*message*)
String *message*;

message Specifies the non-fatal error message to report.

XtWarning routine calls the installed non-fatal error procedure.

¦ Copyright IBM Corp. 1985, 1991

4.17.186 XtWarningMsg

void XtWarningMsg(*name, type, class, default, params, num_params*)
```
      String   name;
      String   type;
      String   class;
      String   default;
      String   *params;
      Cardinal *num_params;
```

name Specifies the general kind of error.

type Specifies the detailed name of the error.

class Specifies the resource class. The Intrinsics internal warnings all have class **XtToolkitError**.

default Specifies the default message to use if an error database entry is not found.

params Specifies a pointer to a list of values to be stored in the message.

num_params Specifies the number of values in the parameter list.

XtWarningMsg routine calls the installed high-level warning handler.

¦ Copyright IBM Corp. 1985, 1991

4.17.187 XtWidgetClassProc

typedef void (*XtWidgetClassProc)(*WidgetClass*);

WidgetClass Specifies the widget class.

XtWidgetClassProc is the type of the *class_part_initialize* procedure of a widget class.

¦ Copyright IBM Corp. 1985, 1991

4.17.188 XtWidgetProc

typedef void (*XtWidgetProc)();

void *WidgetProc(widget)*
Widget *widget*;

widget Specifies the widget.

XtWidgetProc is the destroy procedure type. Destroy procedures are called in subclass-to-superclass order. Therefore, a destroy procedure for a widget should only deallocate storage specific to the subclass and not to its superclasses. The destroy procedure entry in the widget class record is **NULL** if a widget does not need to deallocate any storage. The deallocation of storage includes calling:

XtWidgetProc is also the procedure type for the *insert_child* class routine called by **XtCreateWidget** to add a child to the children array. Other routines calling this type are:

XtFree on dynamic storage allocated with **XtMalloc**, **XtCalloc**, and other allocation procedures

XtRemoveAllCallbacks on callback lists

XtDestroyPixmap on pixmaps allocated with **XtGetPixmap**

XFreePixmap on pixmaps created with direct X calls

XtDestroyGC on **GC**s allocated with **XtGetGC**

XFreeGC on **GC**s allocated with direct X calls

XtRemoveEventHandler on event handlers added with **XtAddEventHandler**

XtRemoveTimeOut on timers created with **XtAddTimeOut**.

¦ Copyright IBM Corp. 1985, 1991

4.17.189 XtWidgetToApplicationContext

XtAppContext XtWidgetToApplicationContext*(widget)*
Widget *w;*;

widget Specifies the widget for which you want the application context.

XtWidgetToApplicationContext gets the application context for a given widget.

It returns the application context for the specified widget.

¦ Copyright IBM Corp. 1985, 1991

4.17.190 XtWindow

Window XtWindow(*widget*)
     Widget *widget*;

widget Specifies the widget.

XtWindow returns the window of the specified widget.

¦ Copyright IBM Corp. 1985, 1991

4.17.191 XtWindowToWidget

Widget XtWindowToWidget(*display, window*)
Display **display*;
Window *window*;

display Specifies the display on which the window is defined.

window Specifies the window for which you want the widget.

XtWindowToWidget routine translates a window and display pointer into a widget instance.

¦ Copyright IBM Corp. 1985, 1991

4.17.192 XtWorkProc

```
typedef Boolean(*XtWorkProc)(caddr_t)
      caddr_ client_data;
```

client_data Specifies the client data generated when the work procedure was registered.

XtWorkProc is the work procedure pointer for the idle-time work routines, **XtAppAddWorkProc** and **XtRemoveWorkProc**.

¦ Copyright IBM Corp. 1985, 1991

5.0 Chapter 5. Protocols

Subtopics

¦ Copyright IBM Corp. 1985, 1991

5.1 CONTENTS

¦ Copyright IBM Corp. 1985, 1991

5.2 About This Chapter

The Core protocol is composed of requests, replies, errors and events. General information about protocol formats, syntax, types, errors, keyboard key assignments, pointers, and predefined atoms is followed by general information about connection setup, connection close, event generation, and flow control.

This chapter also contains a list of each available request with its arguments, return arguments (if necessary), a list of possible error types that can be generated by the request, and a description of the request.

Description of protocol events follows information on requests. Each event is listed with its value, type, and description.

The last section of this chapter contains two lists. The first is an alphabetical listing of **Xlib** functions and corresponding protocol requests. The second lists protocol requests alphabetically with the corresponding **Xlib** functions.

¦ Copyright IBM Corp. 1985, 1991

5.3 Protocol Formats

The formats of protocol requests, replies, errors, and events are explained in this section.

Subtopics
5.3.1 Request Format
5.3.2 Reply Format
5.3.3 Error Format
5.3.4 Event Format

¦ Copyright IBM Corp. 1985, 1991

5.3.1 Request Format

Every protocol request contains an 8-bit major opcode and a 16-bit length field expressed in units of 4 bytes. A protocol request consists of a 4-byte header containing the major opcode, the length field, and a data byte, followed by zero or more additional bytes of data. Unused bytes in a protocol request are not required to be zero.

The length field in the protocol request defines the total length of the protocol request, including the header. The length in a request must equal the minimum length required to contain the request. If the specified length is smaller or larger than the required length, an error is generated.

Extension requests typically have an additional minor opcode encoded in the spare data byte in the request header, but the placement and interpretation of this minor opcode, and all other fields in extension requests, are not defined by the Core protocol. Extensions are intended to contain multiple requests.

Major opcodes 128 through 255 are reserved for extensions.

Every protocol request is implicitly assigned a sequence number starting with one used in replies, errors, and events.

¦ Copyright IBM Corp. 1985, 1991

5.3.2 Reply Format

Every protocol reply contains a 32-bit length field expressed in units of 4 bytes. A protocol reply consists of 32 bytes, followed by zero or additional bytes of data, as specified in the length field. Unused bytes within a protocol reply are not guaranteed to be zero. A protocol reply contains the least significant 16 bits of the sequence number of the corresponding protocol request.

¦ Copyright IBM Corp. 1985, 1991

5.3.3 Error Format

Protocol error reports are 32 bytes long. A protocol error includes an 8-bit error code. Every error includes the major and minor opcodes of the failed request and the least significant 16 bits of the sequence number of the request. Unused bytes within an error are not guaranteed to be zero. Error codes 128 through 255 are reserved for extensions.

The failing resource ID is returned for the following errors:

Colormap

Cursor

Drawable

Font

GContext

IDChoice

Pixmap

Window

The atom that failed is returned for **Atom** errors.

The value that failed is returned for **Value** errors.

Other Core errors return no additional data.

¦ Copyright IBM Corp. 1985, 1991

5.3.4 Event Format

Protocol events are 32 bytes long. Protocol events contain an 8-bit type code. The most significant bit in this code is set if the event is generated by a **SendEvent** request. Unused bytes within a protocol event are not guaranteed to be zero. Event codes 64 through 127 are reserved for extensions, although the Core protocol does not define a mechanism for selecting interest in such events. Every Core event, with the exception of **KeymapNotify**, contains the least significant 16 bits of the sequence number of the last protocol request issued by the client that was, or is currently being, processed by the server.

¦ Copyright IBM Corp. 1985, 1991

5.4 Protocol Syntax

The following conventions are used to help you identify certain components.

Component	**Syntax**
set of alternatives	Appear in **boldface** within braces ({...}).
set of structure components	Appear within brackets ([...]).
types	Appear in UPPERCASE.
alternative values	Appear in Initial Caps.
requests	Appear in this format: **RequestName** *arg1: type1* ... *rgN: typeN* => *result1: type1* ... *resultM: typeM* Errors: **kind2, ..., kindK** Description. The return symbol (**"=>**") in the request code definition indicates the request has one or more replies. If no symbol is present, then the request has no reply and is asynchronous. Errors can still be reported.
events	Appear in this format: **EventName** *value1: type1* ... *valueN: typeN* Description.

¦ Copyright IBM Corp. 1985, 1991

5.5 Common Protocol Types

INT16 and **CARD16** are defined in **Xmd.h** file machine-dependent declarations.

Type	**Description**
ARC	[*x, y*: **INT16** *width, height*: **CARD16** *angle1, angle2*: **INT16**]
ATOM	32-bit value (top 3 bits guaranteed to be zero)
BITGRAVITY	{**Forget, Static,** **NorthWest, North, NorthEast,** **West, Center, East,** **SouthWest, South, SouthEast**}
BITMASK	Various requests contain arguments of the following form: *value-mask*: **BITMASK** These arguments allow the client to specify a subset of a heterogeneous collection of optional arguments. The *value-mask* specifies which arguments are to be provided; each such argument is assigned a unique bit position. The representation of the **BITMASK** typically contains more bits than there are defined arguments; unused bits in the *value-mask* must be zero. Otherwise, the server generates a **Value** error.
BOOL	{**True**, **False**}
BUTMASK	{**Button1, Button2, Button3, Button4, Button5**}
BUTTON	**CARD8**
BYTE	8-bit value
CHAR2B	[*byte1, byte2*: **CARD8**]
CARD8	8-bit unsigned integer
CARD16	16-bit unsigned integer
CARD32	32-bit unsigned integer
COLORMAP	32-bit value (top 3 bits guaranteed to be zero)
CURSOR	32-bit value (top 3 bits guaranteed to be zero)
DEVICEEVENT	{**KeyPress, KeyRelease, ButtonPress, ButtonRelease,** **PointerMotion, Button1Motion, Button2Motion,** **Button3Motion, Button4Motion, Button5Motion,** **ButtonMotion**}

¦ Copyright IBM Corp. 1985, 1991

DRAWABLE	**WINDOW** or **PIXMAP**
Type	**Description**
EVENT	{**KeyPress, KeyRelease, OwnerGrabButton, ButtonPress, ButtonRelease, EnterWindow, LeaveWindow, PointerMotion, PointerMotionHint, Button1Motion, Button2Motion, Button3Motion, Button4Motion, Button5Motion, ButtonMotion, Exposure, VisibilityChange, ResizeRedirect, StructureNotify, SubstructureNotify, SubstructureRedirect, FocusChange, PropertyChange, ColormapChange, KeymapState**}
FONT	32-bit value (top 3 bits guaranteed to be zero)
FONTABLE	**FONT** or **GCONTEXT**
GCONTEXT	32-bit value (top 3 bits guaranteed to be zero)
HOST:	[*family*: {**Internet**} *address*: **LISTofBYTE**] The length, format, and interpretation of a **HOST** address
INT8	8-bit signed integer
INT16	16-bit signed integer
INT32	32-bit signed integer
KEYCODE	**CARD8**
KEYBUTMASK	**KEYMASK** or **BUTMASK**
KEYMASK	{**Shift, Lock, Control, Mod1, Mod2, Mod3, Mod4, Mod5**}
KEYSYM	32-bit value (top 3 bits guaranteed to be zero)
LISTofFOO	A type name in the form of **LISTofFOO** means a counted list of elements of type **FOO**. The size of the length field may vary and is not necessarily the same size as a **FOO**. In some cases, the size may be implicit and not fully specified in this document.
LISTofVALUE	Various requests contain arguments of the following form: *value-list*: LISTofVALUE These arguments allow the client to specify a subset of a heterogeneous collection of *optional* arguments. The *value-list* contains one value for each bit in the mask, from the least to the most significant bit in the mask. Each value is represented with 4 bytes, but the actual value occupies only the least significant bytes as required. The values of the unused bytes do not matter.

¦ Copyright IBM Corp. 1985, 1991

OR	A type of the form **T1 or ... or Tn** means the union of the indicated types; a single-element type is given as the element without enclosing braces.
PIXMAP	32-bit value (top 3 bits guaranteed to be zero)
Type	**Description**
POINT	[*x, y*: **INT16**]
POINTEREVENT	{**ButtonPress, ButtonRelease, EnterWindow, LeaveWindow, PointerMotion, PointerMotionHint, Button1Motion, Button2Motion, Button3Motion, Button4Motion, Button5Motion, ButtonMotion KeymapState**}
RECTANGLE	[*x, y*: **INT16** *width, height*: **CARD16**] The [*x,y*] coordinates of a **RECTANGLE** specify the upper-left corner.
STRING8	**LISTofCARD8**
STRING16	**LISTofCHAR2B** The primary interpretation of *large* characters in a **STRING16** is that these characters are composed of two bytes used to index a 2-D matrix; hence the use of **CHAR2B** rather than **CARD16**. This corresponds to the JIS/ISO method of indexing 2-byte characters. It is expected that most large fonts will be defined with 2-byte matrix indexing. For large fonts constructed with linear indexing, a **CHAR2B** can be interpreted as a 16-bit number by treating *byte1* as the most significant byte.
TIMESTAMP	**CARD32**
VISUALID	32-bit value (top 3 bits guaranteed to be zero)
VALUE	32-bit quantity (used only in **LISTofVALUE**)
WINDOW	32-bit value (top 3 bits guaranteed to be zero)
WINGRAVITY	{**Unmap, Static, NorthWest, North, NorthEast, West, Center, East, SouthWest, South, SouthEast**}

¦ Copyright IBM Corp. 1985, 1991

5.6 Protocol Errors

Error	Cause
Access	One of the following: An attempt to grab a key-button combination already grabbed by another client. An attempt to free a colormap entry not allocated by the client. An attempt to store into a read-only or an unallocated colormap entry. An attempt to modify the access control list from other than the local host (or otherwise authorized client). An attempt to select an event type that, at most, one client can select at a time when another client has already selected it.

Error	Cause
Alloc	The server failed to allocate the requested resource. **Note:** This only covers allocation errors at a very coarse level and is not intended to cover all cases of a server running out of allocation space in the middle of service. The semantics when a server runs out of allocation space are left unspecified.
Atom	A value for an *atom* argument does not name a defined atom.
Colormap	A value for a *colormap* argument does not name a defined colormap.
Cursor	A value for a *cursor* argument does not name a defined cursor.
Drawable	A value for a *drawable* argument does not name a defined window or pixmap.
Font	A value for a *font* argument does not name a defined font.
Fontable	A value for a *fontable* argument does not name a defined font or a defined **gcontext**.
GContext	A value for a *gcontext* argument does not name a defined **gcontext**.
IDChoice	The value chosen for a resource identifier is either not included in the range assigned to the client or is already in use.

¦ Copyright IBM Corp. 1985, 1991

Implementation The server does not implement an aspect of the request. A server that generates this error for a Core request is deficient. As such, this error is not listed for any of the requests, but clients should be prepared to receive such errors and handle or discard them.

Length The length of a request is shorter or longer than that required to minimally contain the arguments.

The length of a request exceeds the maximum length accepted by the server.

Match An **InputOnly** window is used as a drawable.

In a graphics request, the *gcontext* argument does not have the same root and depth as the destination *drawable* argument.

An argument (or pair of arguments) has the correct type and range, but fails to match in some other way required by the request.

Name A font or color of the name specified does not exist.

Pixmap A value for a *pixmap* argument does not name a defined pixmap.

Request The major or minor opcode does not specify a valid request.

Error **Cause**

Value A numeric value falls 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 type is accepted. Any argument defined as a set of alternatives can typically generate this error due to the encoding.

Window A value for a *window* argument does not name a defined window.

Notes:

1. The **Atom, Colormap, Cursor, Drawable, Font, GContext, Pixmap,** and **Window** errors are also used when the argument type is extended by union with a set of fixed alternatives, such as <**Window** or **PointerRoot** or **None**>.

2. In general, when a protocol request terminates with an error, the request has no side effects and there is no partial execution. The requests for which this is *not* true include **ChangeWindowAttributes, ChangeGC, PolyText8, PolyText16, FreeColors, StoreColors,** and **ChangeKeyboardControl**.

¦ Copyright IBM Corp. 1985, 1991

5.7 Changing Keycodes and Keysyms

A keycode represents a physical or logical key. Keycodes lie in the [8,255] inclusive range. A keycode value carries no intrinsic information, although server implementors may attempt to encode geometry (matrix) information interpreted in a server-dependent way. The mapping between keys and keycodes cannot be changed using the protocol.

A keysym is an encoding of a symbol on the keycap. Defined keysyms includes the following character sets:

AP
Arabi
Crylli
Gree
Hebre
Kan
Latin 1, Latin 2, Latin 3, Latin
Publis
Specia
Tec
A set of symbols common on keyboards (such as **RETURN**, **HELP**, and **TAB**).

Keysyms, with the most-significant bit of the 29-bit set, are reserved for vendors.

A list of keysyms is associated with each keycode. The length of this list can vary with each keycode. The list of keysyms conveys the set of symbols on the corresponding key. By convention, if the list contains a single keysym that is alphabetic and case-sensitive, then it should be treated as equivalent to a two-element list including both the lowercase keysyms and the uppercase keysyms. For example, if the list contains a single keysym for uppercase *A*, the client should treat the keysym as if it was two keysyms. One keysym is the lowercase *a* and the other keysym is an uppercase *A*.

For any keycode, the first keysym in the list normally interprets a **KeyPress** when modifier keys are down. The second keysym in the list is interpreted as the **Shift** modifier or the **Lock** modifier is on and **Lock** is interpreted as **ShiftLock**. When the **Lock** modifier is on and is interpreted as **CapsLock**, it is suggested that the **Shift** modifier choose a keysym first. If that keysym is lowercase alphabetic, the corresponding uppercase keysym should be used instead. Other interpretations of **CapsLock** are possible; for example, it can be viewed as equivalent to **ShiftLock**, but only when the first keysym is lowercase alphabetic and the second keysym is the corresponding uppercase alphabetic. No interpretation of keysyms beyond the first two in a list is suggested here. No spatial geometry of the symbols on the key is defined by their order in the keysym list, although a geometry might be defined on a vendor-specific basis.

The mapping between keycodes and keysyms is not used directly by the server, but it is stored for reading and writing by clients.

The *keymask* modifier named **Lock** should be mapped to a **CapsLock** or a **ShiftLock** key. This key assignment is application or user-specific and should be done according to the associated keysyms of the corresponding keycode.

¦ Copyright IBM Corp. 1985, 1991

5.8 Using Predefined Atoms

Predefined atoms are not necessary and may not be useful in all environments, but eliminate many **InternAtom** requests in most applications. Note that *predefined* is used for numeric values, not for required semantics. The Core protocol imposes no semantics on these names, except as they are used in **FONTPROP** structures (see "QueryFont" in topic 5.13.90).

The following names have predefined atom values. (These atom names should be all uppercase.)

ARC	ITALIC_ANGLE	STRING
ATOM	MAX_SPACE	SUBSCRIPT_X
BITMAP	MIN_SPACE	SUBSCRIPT_Y
CAP_HEIGHT	NORM_SPACE	SUPERSCRIPT_X
CARDINAL	NOTICE	SUPERSCRIPT_Y
COLORMAP	PIXMAP	UNDERLINE_POSITION
COPYRIGHT	POINT	UNDERLINE_THICKNESS
CURSOR	POINT_SIZE	VISUALID
CUT_BUFFER0	PRIMARY	WEIGHT
CUT_BUFFER1	QUAD_WIDTH	WINDOW
CUT_BUFFER2	RECTANGLE	WM_CLASS
CUT_BUFFER3	RESOLUTION	WM_CLIENT_MACHINE
CUT_BUFFER4	RESOURCE_MANAGER	WM_COMMAND
CUT_BUFFER5	RGB_BEST_MAP	WM_HINTS
CUT_BUFFER6	RGB_BLUE-MAP	WM_ICON_NAME
CUT_BUFFER7	RGB_COLOR_MAP	WM_ICON_SIZE
DRAWABLE	RGB_DEFAULT_MAP	WM_NAME
END_SPACE	RGB_GRAY_MAP	WM_NORMAL_HINTS
FAMILY_NAME	RGB_GREEN_MAP	WM_SIZE_HINTS
FONT	RGB_RED_MAP	WM_TRANSIENT_FOR
FONT_NAME	SECONDARY	WM_ZOOM_HINTS
FULL_NAME	STRIKEOUT_DESCENT	X_HEIGHT
INTEGER	STRIKEOUT_DESCENT	

Names private to a particular application or end user, but stored in globally accessible locations, should begin with two leading underscores. To avoid conflicts with other names, for which semantics might be imposed (either at the protocol level or in terms of higher level user interface models), names beginning with an underscore should be used for atoms that are private to a particular vendor or organization. To guarantee that conflicts between vendors and organizations are avoided, additional prefixes should be used, but the mechanism for choosing such prefixes is not defined here.

¦ Copyright IBM Corp. 1985, 1991

5.9 Setting Up a Connection

For remote clients, the X-Windows protocol can be built on top of any reliable byte stream. Use the following steps to set up a connection.

Subtopics
5.9.1 Sending Initial Data
5.9.2 Receiving Data

¦ Copyright IBM Corp. 1985, 1991

5.9.1 Sending Initial Data

The client must send an initial byte of data to identify the byt order to be used. The value of the byte must be octal 102 or 154.

The octal 102 (ASCII uppercase B) means values are transmitted with the most-significant byte first. Octal 154 (ASCII lowercase l) means values are transmitted with the least-significant byte first. Except where explicitly noted in the protocol, all 16-bit and 32-bit quantities sent by the client must be transmitted with this byte order, and all 16-bit and 32-bit quantities returned by the server will be transmitted with this byte order.

The following information is transmitted by the client at connectio time after the byte-order byte is transmitted.

Connection information sent by client:

protocol-major-version:	**CARD16**
protocol-minor-version:	**CARD16**
authorization-protocol-name:	**STRING8**
authorization-protocol-data:	**STRING8**

The *version* numbers indicate the version of the protocol the client expects the server to implement.

The *authorization* name indicates the authorization protocol the client expects the server to use and the data specific to that protocol.

Specifying valid authorization mechanisms is not part of the Core protocol. Eventually, one authorization protocol will be agreed upon. In the meantime, a server that implements a protocol other than the protocol the client expects or a server that implements only the host-based mechanism can simply ignore this information. If both name and data strings are empty, this occurrence is to be interpreted as *no explicit authorization.*

¦ Copyright IBM Corp. 1985, 1991

5.9.2 Receiving Data

The client receives the information in the following order.

Connection information received by client:	
success:	**BOOL**
protocol-major-version:	**CARD16**
protocol-minor-version:	**CARD16**
length:	**CARD16**

The *length* is the amount (in units of 4-bytes) of additional data to follow.

The *protocol version* numbers indicate the protocol version supported by the server. The protocol version supported by the server may not equal the version sent by the client. The server can, but does not have to, refuse connections from clients with a version different from the version supported by the server. A server can, but does not have to, support more than one version simultaneously.

The version numbers are for future protocol revisions. Generally, the *major* version would increment for incompatible changes and the *minor* version would increment for small upward compatible changes. If there are no other protocol revisions,

The *major* version is 11.

The *minor* version is 0.

Subtopics

¦ Copyright IBM Corp. 1985, 1991

5.9.2.1 Receiving Additional Data

If authorization fails:

Type	**Options**
reason:	**STRING8**

If authorization is accepted:

Type	**Options**
vendor:	**STRING8**
release-number:	**CARD32**
resource-id-base:	**CARD32**
resource-id-mask:	**CARD32**
image-byte-order:	{**LSBFirst**, **MSBFirst**}
bitmap-scanline-unit:	{8, 16, 32}
bitmap-scanline-pad:	{8, 16, 32}
bitmap-bit-order:	{LeastSignificant, MostSignificant}
pixmap-formats:	**LISTofFORMAT**
roots:	**LISTofSCREEN**
motion-buffer-size:	**CARD32**
maximum-request-length:	**CARD16**
min-keycode:	**KEYCODE**
max-keycode:	**KEYCODE**

where:

FORMAT:

Type		**Options**
	[depth:	**CARD8**,
	bits-per-pixel:	{1, 4, 8, 16, 24, 32}
	scanline-pad:	{8, 16, 32}]

SCREEN:

Type		**Options**
	[root:	**WINDOW**
	width-in-pixels:	**CARD16**
	height-in-pixels:	**CARD16**
	width-in-millimeter	**CARD16**
	height-in-millimete	**CARD16**
	allowed-depths:	**LISTofDEPTH**
	root-depth:	**CARD8**
	root-visual:	**VISUALID**
	default-colormap:	**COLORMAP**
	white-pixel:	**CARD32**
	black-pixel:	**CARD32**
	min-installed-maps:	**CARD16**
	max-installed-maps:	**CARD16**
	backing-stores:	{**Never, WhenMapped, Always**}
	save-unders:	**BOOL**
	current-input-masks	**SETofEVENT**]

DEPTH:

Type		**Options**
	[depth:	**CARD8**

¦ Copyright IBM Corp. 1985, 1991

```
  visuals:              LISTofVISUALTYPE]

VISUALTYPE:

Type                                Options
                    [visual-id:     VISUALID
                    class:          {StaticGray, StaticColor,
                                    TrueColor, GrayScale, PseudoColor,
                                    DirectColor}
                    red-mask:       CARD32
                    green-mask:     CARD32
                    blue-mask:      CARD32
                    bits-per-rgb-value: CARD8
                    colormap-entries:   CARD16]
```

¦ Copyright IBM Corp. 1985, 1991

5.9.2.2 Defining the Server

The following information about the server is provided.

The *vendor string* identifies the owner of the server implementation.

The semantics of the *release-number* is controlled by the vendor.

The *resource-id-mask* contains a single contiguous set of bits (at least 18). The client allocates resource IDs for types **WINDOW, PIXMAP, CURSOR, FONT, GCONTEXT,** and **COLORMAP** by choosing a value with only a subset of these bits set and ORing it with *resource-id-base*. Only values constructed in this way can be used to name newly created resources over this connection. Resource IDs never have the top three bits set. The client is not restricted to linear or contiguous allocation of resource IDs. An ID, once freed, can be reused.

An ID must be unique with respect to the IDs of all other resources, not just other resources of the same type. Note, however, that the value spaces of resource identifiers, atoms, visualids, and keysyms are distinguished by context. As such, they are not required to be disjointed. For example, a given numeric value might be both a valid window ID and a valid atom as well as a valid keysym.

Although the server is, in general, responsible for byte-swapping dat to match the client, images are always transmitted and received in formats (including byte order) specified by the server. The byte order for images is given by *image-byte-order*, and applies to each ***scanline*** unit in **XYFormat** (bitmap) format and to each pixel value in **ZFormat**.

A bitmap is represented in ***scanline order***. Each scanline is padded to a multiple of bits as given by *bitmap-scanline-pad*. The pad bits are of arbitrary value.

The scanline is quantified in multiples of bits as given by *bitmap-scanline-unit*.

Within each unit, the leftmost bit in the bitmap is either the least-significant bit or the most-significant bit in the unit, as given by *bitmap-bit-order*. If a pixmap is represented in **XYFormat**, each plane is represented as a bitmap, and the planes appear from the most-significant to the least-significant in bit order, with no padding between planes.

The *pixmap-formats* contains one entry for each depth value. The entry describes the **ZFormat** used to represent images of that depth. An entry for a depth is included if any screen supports that depth, and all screens supporting that depth must support only that **ZFormat** for that depth. In **ZFormat**, the pixels are in scanline order, left to right, within a scanline.

The number of bits used to hold each pixel is given by *bits-per-pixel*. These may be larger than strictly required by the depth, in which case the least-significant bits hold the pixmap data and the values of the unused high-order bits are undefined. When the *bits-per-pixel* is four, the order of nibbles in the byte is the same as the *image byte-order*. When the *bits-per-pixel* is one, the format is identical for bitmap format. Each scanline is padded to a multiple of bits as given by *scanline-pad*; when *bits-per-pixel* is one, this will be

¦ Copyright IBM Corp. 1985, 1991

identical to *bitmap-scanline-pad*.

The server implementation, which is transparent to the protocol determines how a pointing device moves on the screen. No geometry among screens is defined.

The server can retain the recent history of pointer motion with finer granularity than is reported by **MotionNotify** events. Such history is available by using the **GetMotionEvents** request. The approximate size of the history buffer is given by *motion-buffer-size*.

The *maximum-request-length* specifies in 4-byte units the maximum length of a request that can be accepted by the server. Requests larger than this value generate a **Length** error. If an error is generated, server reads and discards the entire request. The *maximum-request-length* is always at least 4096. Requests, up to and including 16384 bytes in length, are accepted by all servers.

The *min-keycode* specifies the smallest keycode values transmitted by the server. The *min-keycode* is never less than 8.

The *max-keycode* specifies the largest keycode values transmitted by the server. The *max-keycode* is never greater than 255. Not all keycodes in this range are required to have corresponding keys.

¦ Copyright IBM Corp. 1985, 1991

5.9.2.3 Defining the Screen

The following information about the screen is provided.

The *allowed-depths* specifies which pixmap and window depths are supported. Pixmaps are supported for each depth listed. Windows of that depth are supported if at least one visual type is listed for the depth. A pixmap depth of one is always supported and listed, but windows of depth one might not be supported. A depth of zero is never listed, but zero-depth **InputOnly** windows are always supported.

The *root-depth* specifies the depth of the root window.

The *root-visual* specifies the visual type of the root window.

The *width-in-pixels* and *height-in-pixels* specify the size of the root window. The *width-in-pixels* and *height-in-pixels* cannot be changed.

The *class* of the root window is always **InputOutput**.

The *width-in-millimeters* and *height-in-millimeters* can be used to determine the physical size and the aspect ratio.

The *default-colormap* is the colormap initially associated with the root window. Clients with minimal color requirements, that are creating windows of the same depth as the root window, may want to allocate from this colormap by default.

The *black-pixel* and *white-pixel* can be used in implementing a monochrome application. These pixel values are for permanently allocated entries in the *default-colormap*. The actual RGB values can be set on some screens and may not actually be black and white. The names black and white are intended to convey the expected relative intensity of the colors.

The border of the root window is initially a pixmap filled with the *black-pixel*. The initial background of the root window is a pixmap filled with some unspecified two-color pattern using *black-pixel* and *white-pixel*.

The *min-installed-maps* specifies the number of colormaps that can be guaranteed to be installed simultaneously with **InstallColormap**, regardless of the number of entries allocated in each colormap.

The *max-installed-maps* specifies the maximum number of colormaps that can be installed simultaneously, depending on their allocations. Multiple static-visual colormaps with identical contents but different resource IDs should be considered as a single colormap for the purposes of this number. The values for a single hardware colormap is a one.

The *backing-stores* indicates when the server supports backing stores for this screen. However, the backing store might have limited storage for the number of windows it can support simultaneously.

If *save-unders* is**True**, then the server can support the *save-under* mode in **CreateWindow** and **ChangeWindowAttributes**, although again it may be limited storage.

The *current-input-events* is the value returned by **GetWindowAttributes**

¦ Copyright IBM Corp. 1985, 1991

for the *all-event-masks* for the root window.

¦ Copyright IBM Corp. 1985, 1991

5.9.2.4 Defining Visual Type

A visual type can be listed for more than one depth or for more than one screen.

For **PseudoColor**, a pixel value indexes a colormap that produces independent RGB values. The RGB values can be changed dynamically.

For **GrayScale**, a pixel value indexes a colormap that produces independent RGB values, except that the primary that drives the screen is undefined. Consequently, the client should always store the same value for red, green, and blue in colormaps.

For **DirectColor**, a pixel value is decomposed into separate RGB subfields, and each subfield separately indexes the colormap indexes for the corresponding value. The RGB values can be changed dynamically.

The *red-mask*, *green-mask*, and *blue-mask* are defined for **DirectColor**. Each mask has one contiguous set of bits with no intersections. Each mask usually has the same number of one bits.

For **TrueColor**, a pixel value is decomposed into separate RGB subfields, and each subfield separately indexes the colormap indexes for the corresponding value. This colormap has predefined read-only RGB values which are server-dependent, but provide near-linear or linear increasing ramps in each primary.

The *red-mask*, *green-mask*, and *blue-mask* are defined for **TrueColor**. Each mask has one contiguous set of bits with no intersections. Each mask usually has the same number of one bits.

For **StaticColor**, a pixel value indexes a colormap that produces independent RGB values. This colormap has predefined read-only RGB values, which are server-dependent.

For **StaticGray**, a pixel value indexes a colormap that produces independent RGB values. The red, green, and blue values in this colormap are equal for any single pixel value, resulting in shades of gray. **StaticGray** with a 2-entry colormap can be thought of as monochrome.

The *bits-per-rgb-value* specifies the log base 2, the number of distinct color intensity values (individually) of red, green, and blue. This number is not necessarily related to the number of colormap entries. Actual RGB values are always passed in the protocol within a 16-bit spectrum, with zero being minimum intensity and 65535 being the maximum intensity. On hardware that provides a linear zero-based intensity ramp, the following relationship exists:

```
hw-intensity = protocol-intensity / (65536 / total-hw-intensities)
```

The *colormap-entries* defines the number of colormap entries available in a newly created colormap. Colormap entries are indexed from zero. For **DirectColor** and **TrueColor**, this is usually two to the power of the maximum number of one bits in *red-mask*, *green-mask*, and *blue-mask*.

¦ Copyright IBM Corp. 1985, 1991

5.10 Closing Connections to the Server

At the time of ***connection close***, all event selections made by the client are discarded in the following ways:

An **UngrabPointer** is performed, if the client has the pointer actively grabbed.

An **UngrabKeyboard** is performed releasing all passive grabs by the client, if the client has the keyboard actively grabbed.

An **UngrabServer** is performed disowning all selections owned by the client, if the client has the server grabbed. See "SetSelectionOwner" in topic 5.13.108 for more information.

If close-down mode (see "SetCloseDownMode" in topic 5.13.101) is **RetainPermanent** or **RetainTemporary**, then all resources, including colormap entries, allocated by the client are marked as permanent or temporary. This marking does not prevent other clients from destroying these resources explicitly.

If the mode is **Destroy**, all client resources are destroyed.

When the resources of a client and the members in the client save-set, that is an inferior of a window created by the client, are destroyed, the save-set window is reparented to the closest ancestor so that the save-set window is not an inferior of a window created by the client. If the save-set window is unmapped, a **MapWindow** request is performed on it, even if it is not an inferior of a window created by the client. After save-set processing, all windows created by the client are destroyed. For each non-window resource created by the client, the appropriate **Free** request is performed. All colors and colormap entries allocated by the client are freed.

When the last connection to a server closes, a server goes through a cycle of having no connections and having some connections. The server resets its state to its initial state as if it had just been started, with each transition of the state having no connections (that is, the **Destroy** close-down mode forced a connection closing). The server starts the cycle by destroying all lingering resources of clients that were terminated in **RetainPermanent** or **RetainTemporary** mode. The X Server does the following:

Deletes everything except the predefined atom identifier
Deletes all the properties on all root window
Resets all device maps and attributes (key click, bell volume acceleration) to initial value
Resets the access control list to initial valu
Restores the standard root tiles and cursors to initial valu
Restores the default font path to initial valu
Restores the input focus to state **PointerRoot**.

Closing a connection with a close-down mode of **RetainPermanent** or **RetainTemporary** does not reset the server.

¦ Copyright IBM Corp. 1985, 1991

5.11 Generating an Event

When a button is pressed with the pointer in window ***WINDOW*** and no active pointer grab is in progress, then the ancestors of ***WINDOW*** are searched from the root down for a passive grab to activate. If a matching passive grab on the button does not exist, then an active grab is started automatically for the client receiving the event, and the *last-pointer-grab* time is set to the current server time. The effect is equivalent to a **GrabButton** with the following arguments:

Argument	**Value**
event-window	The event window.
event-mask	Selected pointer events of the client on the event window.
pointer-mode	**Asynchronous**.
keyboard-mode	**Asynchronous**.
owner-events	**True** if the client has **OwnerGrabButton** selected on the event window. Otherwise, it is set to **False**.
confine-to	None.
cursor	None.

The grab is terminated automatically when all the buttons are released. **UngrabPointer** and **ChangeActiveGrab** can both be used to modify the active grab.

¦ Copyright IBM Corp. 1985, 1991

5.12 Controlling Flow and Concurrency

When the server is writing to a specific connection, it can stop reading from that connection. However, if the writing stops, the server must continue to service other connections. The server is not required to buffer more than a single request per connection at one time. For a given connection to the server, a client can block while reading from the connection, but should undertake to read (events and errors) when writing would block. Failure on the part of a client to do this could result in a deadlocked connection, although deadlock is unlikely unless the transport layer has very little buffering, or the client attempts to send large numbers of requests without reading replies or checking for errors and events.

If a server is implemented with internal concurrency, the overall effect must be that individual requests are executed to completion in a serial order, and that requests from a given connection are executed in delivery order. In other words, the total execution order is a shuffle of the individual streams. The execution of a request includes validating all arguments, collecting all data for any reply, and generating (and queueing) all required events, but does not include the actual transmission of the reply and the events. In addition, the effect of any other cause, such as the activation of a pointer motion that can generate multiple events, must effectively generate (and queue) all required events indivisibly with respect to all other causes and requests.

¦ Copyright IBM Corp. 1985, 1991

5.13 Protocol Requests

This section contains detailed descriptions of the X-Windows protocol requests, which are arranged in alphabetical order.

Subtopics

¦ Copyright IBM Corp. 1985, 1991

5.13.55 GetWindowAttributes
5.13.56 GrabButton
5.13.57 GrabKey
5.13.58 GrabKeyboard
5.13.59 GrabPointer
5.13.60 GrabServer
5.13.61 ImageText16
5.13.62 ImageText8
5.13.63 InstallColormap
5.13.64 InternAtom
5.13.65 KillClient
5.13.66 ListExtensions
5.13.67 ListFonts
5.13.68 ListFontsWithInfo
5.13.69 ListHosts
5.13.70 ListInstalledColormaps
5.13.71 ListProperties
5.13.72 LookupColor
5.13.73 MapSubwindows
5.13.74 MapWindow
5.13.75 NoOperation
5.13.76 OpenFont
5.13.77 PolyArc
5.13.78 PolyFillArc
5.13.79 PolyFillRectangle
5.13.80 PolyPoint
5.13.81 PolyLine
5.13.82 PolyRectangle
5.13.83 PolySegment
5.13.84 PolyText16
5.13.85 PolyText8
5.13.86 PutImage
5.13.87 QueryBestSize
5.13.88 QueryColors
5.13.89 QueryExtension
5.13.90 QueryFont
5.13.91 QueryKeymap
5.13.92 QueryPointer
5.13.93 QueryTextExtents
5.13.94 QueryTree
5.13.95 RecolorCursor
5.13.96 ReparentWindow
5.13.97 RotateProperties
5.13.98 SendEvent
5.13.99 SetAccessControl
5.13.100 SetClipRectangles
5.13.101 SetCloseDownMode
5.13.102 SetDashes
5.13.103 SetFontPath
5.13.104 SetInputFocus
5.13.105 SetModifierMapping
5.13.106 SetPointerMapping
5.13.107 SetScreenSaver
5.13.108 SetSelectionOwner
5.13.109 StoreColors
5.13.110 StoreNamedColor
5.13.111 TranslateCoordinates
5.13.112 UngrabButton
5.13.113 UngrabKey
5.13.114 UngrabKeyboard

¦ Copyright IBM Corp. 1985, 1991

¦ Copyright IBM Corp. 1985, 1991

5.13.1 AllocColor

```
   cmap: COLORMAP
   red, green, blue: CARD16
=>
   pixel: CARD32
   red, green, blue: CARD16

      Errors: Colormap, Alloc
```

AllocColor allocates a read-only colormap entry and returns the pixel value corresponding to the closest available RGB values supported by the hardware. It returns the RGB value actually used.

¦ Copyright IBM Corp. 1985, 1991

5.13.2 AllocColorCells

cmap: **COLORMAP**
colors, planes: **CARD16**
contiguous: **BOOL**
=>
pixels, masks: **LISTofCARD32**

Errors: **Colormap, Value, Alloc**

AlloColorCells allocates color cells. The number of colors must be positive and the number planes must be non-negative. It combines *masks* and *pixels* to produce distinct pixels. The RGB values of the allocated entries are undefined.

If *C* colors and *P planes* are requested, then *C pixels* and *P masks* are returned. No mask will have any bits in common with any other *mask*, or with any of the *pixels*. By ORing the *masks* and *pixels*, **C*2(P)** distinct pixels values can be produced. These pixel values are allocated writable by the request.

If contiguous is **True** and all masks are ORed together, the following will be formed:

A single contiguous set of bits for **GrayScale** or **PseudoColor**
Three contiguous sets of bits (one within each pixel subfield) fo **DirectColor**.

A **Value** error occurs if the number of colors is non-negative or the number of planes is positive.

¦ Copyright IBM Corp. 1985, 1991

5.13.3 AllocColorPlanes

```
    cmap: COLORMAP
    colors, reds, greens, blues: CARD16
    contiguous: BOOL
=>
    pixel: LISTofCARD32
    red-mask, green-mask, blue-mask: CARD32

      Errors: Colormap, Value, Alloc
```

AllocColorPlanes brings together *masks* and *pixels* to produce distinct pixels. The number of colors must be positive and the number planes must be non-negative. The RGB values of the allocated entries are undefined.

If *C* colors, *R* reds, *G* greens, and *B* blues are requested, then *C pixels* are returned, and the masks have R, G, and B bits set respectively.

If *contiguous* is **True**, then each mask will have a contiguous set of bits. No mask will have any bits in common with any other mask, or with any of the *pixels*.

For **DirectColor**, each mask will lie within the corresponding *pixel* subfield. By ORing subsets of masks with *pixels*, **C*2(R+G+B)** distinct *pixels* can be produced; these masks are allocated by the request.

There are only **C*2(R)** independent red entries, **C*2(G)** independent green entries, and **C*2(B)** independent blue entries in the colormaps. This is true even for **PseudoColor**.

When the colormap entry for a pixel value is changed using **StoreColors** or **StoreNamedColor**, the *pixel* is decomposed according to the masks and the corresponding independent entries are updated.

A **Value** error occurs if the number of colors is non-negative or the reds, greens, and blues are positive.

¦ Copyright IBM Corp. 1985, 1991

5.13.4 AllocNamedColor

```
    cmap: COLORMAP
    name: STRING8
=>
    pixel CARD32
    exact-red, exact-green, exact-blue: CARD16
    visual-red, visual-green, visual-blue: CARD16

      Errors: Colormap, Name, Alloc
```

AllocNamedColor searches for the named color of the screen associated with the specified colormap. Then, it completes an **AllocColor** request on *cmap*. The color name in **AllocNamedColor** is case-sensitive.

The exact RGB values specify the true values for the color and the visual values specify the values used in the colormap.

¦ Copyright IBM Corp. 1985, 1991

5.13.5 AllowEvents

mode: {**AsyncPointer, SyncPointer, ReplayPointer, AsyncKeyboard, SyncKeyboard, ReplayKeyboard, AsyncBoth, SyncBoth**}
time: **TIMESTAMP** or **CurrentTime**

Errors: **Value**

AllowEvents releases some queued events if the client has caused a device to freeze. This request has no effect if the specified *time* is earlier than the last-grab time of the most recent active grab for the client or if the specified *time* is later than the current server time.

It is possible for both a pointer grab and a keyboard grab to be active simultaneously by the same clients or by different clients. When a device is frozen on behalf of a pointer grab or a keyboard grab, no event processing is performed for the device. It is possible for a single device to be frozen due to both grabs. In this case, the freeze must be released on behalf of both grabs before the events can be processed again.

If *mode* is **AsyncPointer** and the pointer is frozen by the client, then the pointer event processing continues normally. If the pointer is frozen twice by the client on behalf of two separate grabs, **AsyncPointer** releases both.

AsyncPointer has no effect if the pointer is not frozen by the client, but the pointer does not have to be grabbed by the client.

If *mode* is **SyncPointer** and the pointer is frozen and actively grabbed by the client, then the pointer event processing continues normally until the next **ButtonPress** or **ButtonRelease** event is reported to the client, at which time the pointer again appears to freeze. However, if the reported event causes the pointer grab to be released, then the pointer does not freeze.

SyncPointer has no effect if the pointer is not frozen by the client or if the pointer is not grabbed by the client.

If *mode* is **ReplayPointer** and the pointer is actively grabbed by the client or frozen as the result of an event having been sent to the client (either by a **GrabButton** event or a previous **AllowEvents** with *mode* **SyncPointer**, but not from a **GrabPointer**). Then, the pointer grab is released, the event is completely reprocessed, and the event ignores passive grabs at or above (towards the root) the grab-window of the grab just released.

ReplayPointer has no effect if the pointer is not grabbed by the client or if the pointer is not frozen as the result of an event.

If *mode* is **AsyncKeyboard** and the keyboard is frozen by the client, then keyboard event processing continues normally. If the keyboard is frozen twice by the client on behalf of two separate grabs, **AsyncKeyboard** releases both.

AsyncKeyboard has no effect if the keyboard is not frozen by the client, but the keyboard does not need to be grabbed by the client.

If *mode* is **SyncKeyboard**, and the keyboard is frozen and actively grabbed by the client, keyboard event processing continues normally until the next **KeyPress** or **KeyRelease** event is reported to the client,

¦ Copyright IBM Corp. 1985, 1991

at which time the keyboard appears to freeze again. However, if the reported event causes the keyboard grab to be released, then the keyboard does not freeze.

SyncKeyboard has no effect if the keyboard is not frozen by the client or if the keyboard is not grabbed by the client.

If *mode* is **ReplayKeyboard** and the keyboard is actively grabbed by the client, frozen as the result of an event having been sent to the client (either from a **GrabKey**, or from a previous **AllowEvents** with *mode* **SyncKeyboard**, but not from a **GrabKeyboard**). Then, the keyboard grab is released, the event is completely reprocessed, and the event ignores passive grabs at or above (towards the root) the grab-window of the grab just released.

ReplayKeyboard has no effect if the keyboard is not grabbed by the client, or if the keyboard is not frozen as the result of an event.

If *mode* is **SyncBoth**, and both pointer and keyboard are frozen by the client, then, event processing for both devices continues normally until the next **ButtonPress**, **ButtonRelease**, **KeyPress**, or **KeyRelease** event is reported to the client for a grabbed device (button event for the pointer, key event for the keyboard). At this time, the devices again appear to freeze, unless the reported event causes the grab to be released for both devices and both devices do not freeze or the other device is still grabbed a subsequent event for it cause both devices to freeze.

SyncBoth has no effect unless both pointer and keyboard are frozen by the client. If the pointer of keyboard is frozen twice by the client on behalf of two separate grabs, **SyncBoth** releases both (but a subsequent freeze for **SyncBoth** freezes each device only once).

If *mode* is **AsyncBoth** and both the pointer and the keyboard are frozen by the client, then event processing for both devices continues normally. If a device is frozen twice by the client on behalf of two separate grabs, **AsyncBoth** releases both.

AsyncBoth has no effect unless both pointer and keyboard are frozen by the client.

AsyncPointer, **SyncPointer**, and **Replay Pointer** have no effect on processing of keyboard events. **AsyncKeyboard**, **SyncKeyboard**, and **ReplayKeyboard** have no effect on processing of pointer events.

¦ Copyright IBM Corp. 1985, 1991

5.13.6 Bell

percent: **INT8**

Errors: **Value**

Bell rings the bell on the keyboard at a volume relative to the base volume for the keyboard, if possible. The *percent* can range from -100 to 100 inclusive. The volume at which the bell is rung when *percent* is non-negative is:

```
base - [(base * percent) / 100] + percent
```

and when *percent* is negative:

```
base + [(base * percent) / 100]
```

¦ Copyright IBM Corp. 1985, 1991

5.13.7 ChangeActivePointerGrab

event-mask: **SETofPOINTEREVENT**
cursor: **CURSOR** or **None**
time: **TIMESTAMP** or **CurrentTime**

Errors: **Cursor**

ChangeActivePointerGrab changes the specified dynamic parameters if the pointer is actively grabbed by the client and the specified *time* is no earlier than the last-pointer-grab time and no later than the current server time. This request has no effect on the passive parameters of a **GrabButton**.

If a *cursor* is specified, it is displayed regardless of which window contains the pointer. If a *cursor* is not specified, when the pointer is in *grab-window* or one of its subwindows, the normal cursor for that window is displayed. Otherwise, the *cursor* for the *grab-window* is displayed.

ChangeActivePointerGrab generates events if the *event-mask* specified is in the **Set of Event** masks.

¦ Copyright IBM Corp. 1985, 1991

5.13.8 ChangeGC

gc: **GCONTEXT**
value-mask: **BITMASK**
value-list: **LISTofVALUE**

Errors: **GContext, Pixmap, Font, Match, Value, Alloc**

ChangeGC changes components in **gc**. The *value-mask* and *value-list* specify which components are to be changed. See **CreateGC**, in this chapter, for the values and restrictions.

Changing the *clip-mask* overrides any previous **SetClipRectangles** request on the context. Changing *dash-offset* or *dashes* overrides any previous **SetDashes** request on the context.

The order in which components are verified and altered is server-dependent. If an error is generated, a subset of the components may have been altered.

¦ Copyright IBM Corp. 1985, 1991

5.13.9 ChangeHosts

mode: {**Insert, Delete**}
host: **HOST**

Errors: **Access, Value**

ChangeHosts adds or removes the specified host from the access control list. When access control is enabled and a host attempts to establish a connection to the server, the host must be in this list or the server will refuse the connection.

The client and the host must reside on the same server or the client must have permission by a server-dependent method to execute this request. Otherwise, an **Access** error is returned.

An initial access control list can usually be specified by naming a file that the server reads at startup and reset.

Some address families are defined, but the server may support families that are not defined.

For Class A address, the network number is the first byte in th address, and the host number is the remaining three bytes with the most-significant byte first.

For Class B address, the network number is the first 2 bytes and th host number is the last 2 bytes with the most-significant byte first.

For Class C address, the network number is the first 3 bytes with th most-significant byte first. The last byte is the host number.

Use of an unsupported family or an improper address format or length within a supported family results in a **Value** error.

For the Internet family, the address must be 4 bytes long. The address bytes are in standard Internet Protocol order. The server performs no automatic swapping on the address bytes.

¦ Copyright IBM Corp. 1985, 1991

5.13.10 ChangeKeyboardControl

value-mask: **BITMASK**
value-list: **LISTofVALUE**

Errors: **Match, Value**

ChangeKeyboardControl controls various aspects of the keyboard. The *value-mask* and *value-list* specify which controls are to be changed. The possible values for *value-mask* and *value-list* are:

key-click-percent: **INT8**
bell-percent: **INT8**
bell-pitch: **INT16**
bell-duration: **INT16**
led: **CARD8**
led-mode: {**On, Off**}
key: **KEYCODE**
auto-repeat-mode: {**On, Off, Default**}

The *key-click-percent* value sets the volume for key clicks between 0 (off) and 100 (loud) inclusive. To restore the default, set to **-1**. Other negative values generate a **Value** error.

The *bell-percent* value sets the base volume for the bell between 0 (off) and 100 (loud) inclusive. To restore the default, set to **-1**. Other negative values generate a **Value** error.

The *bell-pitch* value sets the pitch (in Hz) of the bell. To restore the default set to **-1**. Other negative values generate a **Value** error.

The *bell-duration* value sets the duration (in milliseconds) of the bell. To restore the default, set to **-1**. Other negative values generate a **Value** error.

No standard interpretation of LEDs is defined. Numbering from one, 32 LEDs are supported.

If both *led-mode* and *led* are specified, the state of that LED is changed.

If only *led-mode* is specified, the state of all LEDs is changed.

If an LED is specified without an *led-mode*, a **Match** error is returned.

If both *auto-repeat-mode* and *key* are specified, the *auto-repeat* mode of that key is changed. If only *auto-repeat-mode* is specified, the global *auto-repeat* mode for the entire keyboard is changed without affecting the per-key settings.

If *key* is specified without an *auto-repeat-mode*, a **Match** error is returned.

A bell generator, which is connected with the console, but not directly to the keyboard, is treated as if it were part of the keyboard.

The order in which controls are verified and altered is server-dependent. If an error is generated, a subset of the controls may have been altered.

¦ Copyright IBM Corp. 1985, 1991

5.13.11 ChangeKeyboardMapping

first-keycode: **KEYCODE**
keysyms-per-keycode: **CARD8**
keysyms: **LISTofKEYSYM**

Errors: **Value, Alloc**

ChangeKeyboardMapping defines the symbols for the specified number of keycodes, starting with the specified keycode. The symbols for keycodes outside this range remained unchanged. This request generates a **MappingNotify** event. (See "Changing Keycodes and Keysyms" in topic 5.7.)

The number of elements in the *keysyms* list must be a multiple of *keysyms-per-keycode*, otherwise a **Length** error is returned.

The *first-keycode* must be greater than or equal to *min-keycode* as returned in the connection setup, and

```
first-keycode + (keysyms-length/keysyms-per-keycode) -1
```

must be less than or equal to *max-keycode* as returned in the connection setup. The **KEYSYM** number **N** (counting from zero) for *keycode* **K** has an index (counting from zero) of

```
(K - first-keycode) * keysyms-per-keycode + N
```

in *keysyms*.

The *keysyms-per-keycode* value can be chosen arbitrarily by the client to be large enough to hold the necessary symbols. A special **KEYSYM** value of **NoSymbol** should be used for unused elements of individual keycodes. **NoSymbol** can be used in non-trailing positions of the effective list for a keycode.

The server does not have to interpret this mapping; it merely stores it for reading and writing by clients.

¦ Copyright IBM Corp. 1985, 1991

5.13.12 ChangePointerControl

do-acceleration, do-threshold: **BOOL**
acceleration-numerator, acceleration-denominator: **INT16**
threshold: **INT16**

Errors: **Value**

ChangePointerControl defines how the pointer moves. The acceleration is a multiplier for movement. Acceleration is expressed as a fraction. For example, specifying **3/1** means that the pointer moves three times as fast as normal. The fraction can be rounded off arbitrarily by the server.

Acceleration only takes effect if the pointer moves more than *threshold* pixels at once, and only applies to the amount beyond the *threshold*. To restore the default, set to **-1**. Other negative values generate a **Value** error. A zero value for *acceleration-denominator* also generates a **Value** error.

¦ Copyright IBM Corp. 1985, 1991

5.13.13 ChangeProperty

window: **WINDOW**
property, type: **ATOM**
format: {**8, 16, 32**}
mode: {**Replace, Prepend, Append**}
data: **LISTofINT8** or **LISTofINT16** or **LISTofINT32**

Errors: **Window, Atom, Value, Match, Alloc**

ChangeProperty alters the property for the specified window. It generates a **PropertyNotify** event on the window.

The *type* is uninterpreted by the server.

The *format* specifies whether the data should be viewed as a list of 8-bit, 16-bit, or 32-bit quantities so that the server can swap bytes as necessary.

If *mode* is:

- **Replace**, the previous *property* value is discarded.
- **Prepend** or **Append**, the *type* and *format* must match the existing *property* value.

If the *property* is undefined, it is treated as defined with the correct *type* and *format* with zero-length data.

- For **Prepend**, the data is added to the beginning of the existing data.
- For **Append**, the data is added to the end of the existing data.

The maximum size of a property is server-dependent. The lifetime of a property is not tied to the storing client. Properties remain until explicitly deleted, the window is destroyed, or the server is reset. See "Closing Connections to the Server" in topic 5.10.

¦ Copyright IBM Corp. 1985, 1991

5.13.14 ChangeSaveSet

window: **WINDOW**
mode: {**Insert, Delete**}

Errors: **Window, Match, Value**

ChangeSaveSet adds or removes the specified window from the client save-set. The specified window must be created by another client. The server automatically removes windows from the save-set when the client is destroyed. See "Closing Connections to the Server" in topic 5.10.

¦ Copyright IBM Corp. 1985, 1991

5.13.15 ChangeWindowAttributes

window	**WINDOW**
value-mask	**BITMASK**
value-list	**LISTofVALUE**

Errors: **Window, Pixmap, Colormap, Cursor, Match, Value, Access**

ChangeWindowAttributes sets a new background with background-pixmap or background-pixel, and overrides any previous background. Setting a new border with border-pixel or border-pixmap overrides any previous border.

The *value-mask* and *value-list* specify which attributes to change. See "CreateWindow" in topic 5.13.30 for the values and and restrictions.

Changing the background does not cause the window contents to b changed.

Setting the border, or changing the background so that the border til origin changes, causes the border to be repainted.

Changing the background of a root window to **None** or **ParentRelative** restores the default background pixmap.

Changing the border of a root window to **CopyFromParent** restores the default border pixmap.

Changing the win-gravity does not affect the current position of th window.

Changing the backing store of an obscured window to **WhenMapped** or **Always** may have no immediate effect.

Changing the backing-planes, backing-pixel, or save-under of a mappe window may have no immediate effect.

Multiple clients can select input on the same window, but thei *event-masks* are disjointed.

When an event is generated, it is reported to all interested clients However, only one client at a time can select the following:

- **SubstructureRedirect**
- **ResizeRedirect**
- **ButtonPress**.

There is only one *do-not-propagate-mask* per window not one per client.

Changing the *colormap* of a window, by defining a new map, or not changing the contents of the existing map, generates a **ColormapNotify** event.

Changing the *colormap* of a visible window may have no immediate effect on the screen.

Changing the *cursor* of a root window to **None** restores the default cursor.

The order in which attributes are verified and altered is server-dependent. If an error is generated, a subset of the attributes

¦ Copyright IBM Corp. 1985, 1991

```
may have been altered.
```

¦ Copyright IBM Corp. 1985, 1991

5.13.16 CirculateWindow

window: **WINDOW**
direction: {**RaiseLowest, LowerHighest**}

Errors: **Window, Value**

CirculateWindow circulates the specified window in the direction specified. If *direction* is:

RaiseLowest, it raises the lowest mapped child, if any, that is occluded by another child, to the top of the stack.

LowerHighest, it lowers the highest mapped child, if any, that occludes another child, to the bottom of the stack.

CirculateWindow generates a **CirculateNotify** event if the window is restacked. It generates a **CirculateRequest** event if another client selects **SubstructureRedirect** on the window. In this case, no further processing is performed.

Exposure processing is performed on formerly obscured windows.

¦ Copyright IBM Corp. 1985, 1991

5.13.17 ClearArea

window: **WINDOW**
x, y: **INT16**
width, height: **CARD16**
exposures: **BOOL**

Errors: **Window, Value, Match**

ClearArea clears the area within a window.

The *x* and *y* coordinates, which are relative to the origin of the specified window, specify the upper-left corner of the rectangle.

If *width* is zero, it is replaced with the current *width* of the *window* minus *x*.

If *height* is zero, it is replaced with the current *height* of the *window* minus *y*.

In the following cases, if *exposures* is **True**, one or more exposure events for regions of the rectangle that are visible or in a backing store are generated.

If the *window* has a defined background tile, the rectangle is tiled with a plane-mask of all ones and has a function of **Copy**.

If the *window* has background **None**, the contents of the *window* are not changed.

ClearArea cannot be used with an **InputOnly** window.

¦ Copyright IBM Corp. 1985, 1991

5.13.18 CloseFont

font: **FONT**

Errors: **Font**

CloseFont deletes the association between the resource ID and the font. The font is freed when no other resource references it.

¦ Copyright IBM Corp. 1985, 1991

5.13.19 ConfigureWindow

window: **WINDOW**
value-mask: **BITMASK**
value-list: **LISTofVALUE**

Errors: **Window, Match, Value**

ConfigureWindow reconfigures the size, position, border, and stacking order of a window. (Attempts to configure a root window have no effect.) The values to be changed are in *value-mask* and *value-list*. The values not specified are taken from the existing geometry of the window. Possible values for *value-list* are:

x: **INT16**
y: **INT16**
width: **CARD16**
height: **CARD16**
border-width: **CARD16**
sibling: **WINDOW**
stack-mode: {**Above, Below, TopIf, BottomIf, Opposite**}

The *x* and *y* coordinates, which are relative to the origin of the parent window, specify the position of the upper-left outer corner of the window.

The *width* and *height* specify the inside of the window, excluding the border. The *width* must be a non-zero.

Changing just the *border-width* leaves the outer-left corner of the window in a fixed position, but moves the absolute position of the origin of the window. If the *border-width* of an **InputOnly** window is a non-zero, a **Match** error is generated.

If *override-redirect* is **False** and another client has selected **SubstructureRedirect** on the parent, then a **ConfigureRequest** event is generated, and no further processing is performed.

If another client selects **ResizeRedirect** on the window and the inside width or height of the window is being changed, then a **ResizeRequest** event is generated, and the current inside width and height are used instead.

The *override-redirect* attribute of the window has no effect on **ResizeRedirect**. **SubstructureRedirect** on the parent window has precedence over **ResizeRedirect** on the window.

The geometry of the window is changed as specified and the window is restacked among siblings, and a **ConfigureNotify** event is generated.

If a *sibling* and a *stack-mode* are specified, the window is restacked as follows:

Stack-mode	**Stack Order**
Above	Window is placed just above the sibling.
Below	Window is placed just below the sibling.

¦ Copyright IBM Corp. 1985, 1991

TopIf If the sibling occludes the window, the window is placed at the top of the stack.

BottomIf If the window occludes the sibling, the window is placed at the bottom of the stack.

Opposite If the sibling occludes the window, the window is placed at the top of the stack. If the window occludes the sibling, the window is placed at the bottom of the stack.

If a *stack-mode* is specified, but no *sibling* is specified, the window is restacked as follows:

Stack-mode	**Stack Order**
Above	The window is placed at the top of the stack.
Below	The window is placed at the bottom of the stack.
TopIf	If any sibling occludes the window, the window is placed at the top of the stack.
BottomIf	If the window occludes any sibling, the window is placed at the bottom of the stack.
Opposite	If any sibling occludes the window, the window is placed at the top of the stack. If the window occludes any sibling, the window is placed at the bottom of the stack.

If a sibling is specified without a *stack-mode* or if the window is not actually a sibling, a **Match** error occurs.

The computations for **BottomIf**, **TopIf**, and **Opposite** are performed with respect to the final geometry of the window (as controlled by the other arguments to the request), not by the initial geometry of the window.

If the inside *width* or *height* of the window is changed, the children of the window are affected as follows:

- Exposure processing is performed on formerly obscured windows, including the window and its inferiors, if regions of the formerly obscured windows were obscured, but are no longer obscured.

- Exposure processing is also performed on any new regions of the window, as a result of increasing the width or height, and any regions where window contents are lost.

If the inside width or height of a window is not changed, but th window is moved or its border is changed, then the contents of the window are not lost, but moved with the window. Changing the inside width or height of the window causes its contents to be moved or lost, depending on the bit-gravity of the window, and causes children to be reconfigured, depending on their win-gravity.

The following values are for *width* and *height* as [*x, y*] pairs:

NorthWest: [0,0]

¦ Copyright IBM Corp. 1985, 1991

North:	[W/2,0]
NorthEast:	[W,0]
West:	[0,H/2]
Center:	[W/2,H/2]
East:	[W,H/2]
SouthWest:	[0,H]
South:	[W/2,H]
SouthEast:	[W,H]

When a window with one of these bit-gravities is resized, the corresponding pair defines the change in position of each pixel in the window. When a window with one of these win-gravities has its parent window resized, the corresponding pair defines the change in position of the window within the parent. When a window's position is changed, a **GravityNotify** event is generated. **GravityNotify** events are generated after the **ConfigureNotify** event is generated.

A gravity of **Static** indicates that the contents or origin should not move relative to the origin of the root window. If the change in size of the window is coupled with a change in position of [*x, y*], then for *bit-gravity*, the change in position of each pixel is [*-x, -y*], and for *win-gravity*, the change in position of a child when its parent is resized is [*x, y*]. **Static** gravity takes effect only when the width or height of the window is changed, not when the window is moved only.

A *bit-gravity* of **Forget** indicates the following (even if backing store or save-under has been requested):

The window contents are always discarded after a size chang

The window is tiled with its background if no background is defined the existing screen contents are not altered

One or more exposure events are generated

A server can ignore the specified *bit-gravity* and use **Forget** instead.

A *win-gravity* of **Unmap** is like **NorthWest**, except that the child is also unmapped when the parent is resized, and an **UnmapNotify** event is generated. **UnmapNotify** events are generated after the **ConfigureNotify** event is generated.

¦ Copyright IBM Corp. 1985, 1991

5.13.20 ConvertSelection

selection, target: **ATOM**
property: **ATOM** or **None**
requestor: **WINDOW**
time: **TIMESTAMP** or **CurrentTime**

Errors: **Atom, Window**

ConvertSelection converts a selection by sending a request to its owner. The arguments remain unchanged from their original state. This protocol does not change the property.

If the specified *selection* has an owner, the server sends a **SelectionRequest** event to that owner. If there is no owner for the specified *selection*, the server generates a **SelectionNotify** event to the requestor with property **None**.

¦ Copyright IBM Corp. 1985, 1991

5.13.21 CopyArea

src-drawable, dst-drawable: **DRAWABLE**
gc: **GCONTEXT**
x, src-y: **INT16**
width, height: **CARD16**
dst-x, dst-y: **INT16**

Errors: **Drawable, GContext, Match**

CopyArea combines the specified *src-drawable* rectangle with the specified *dst-drawable* rectangle. The source drawable and the destination drawable must have the same root and the same depth. Otherwise, a **Match** error occurs.

The *src-x* and *src-y* coordinates, which are relative to the origin of the source drawable, specify the upper-left corner of the rectangle.

The *dst-x* and *dst-y* coordinates, which are relative to the origin of the destination drawable, specify the upper-left corner of the rectangle.

If regions of the source rectangle are obscured, and have not been retained in backing store, or if regions outside the boundaries of the *src-drawable* are specified, then those regions are not copied. The following occurs on all corresponding destination regions that are visible or retained in backing store.

If the *dst-drawable* is a window with a background other than **None**, the corresponding destination regions are tiled with that background. Regardless of tiling, if graphics-exposures in *gc* is **True**. Then, **GraphicsExposure** events for all corresponding destination regions are generated.

If graphics-exposures is **True**, but no regions are exposed, then a **NoExposure** event is generated.

CopyArea uses the gc components: *function, plane-mask, subwindow-mode, graphics-exposures, clip-x-origin, clip-y-origin*, and *clip-mask*.

¦ Copyright IBM Corp. 1985, 1991

5.13.22 CopyColormapAndFree

mid, src-cmap: **COLORMAP**

Errors: **IDChoice, Colormap, Alloc**

CopyColormapAndFree creates a colormap of the same visual type and for the same screen as specified in the *src-cmap*. It associates the identifier *mid* with the type.

CopyColormapAndFree moves all of the existing allocations of the client from *src-cmap* to the new colormap with the color values intact, and their read-only or writable characteristics intact. It also frees these colormap entries in *src-cmap*. Color values in other entries in the new colormap are undefined.

If *src-cmap* was created by the client with *alloc* **All** (see "CreateColormap" in topic 5.13.25), then the following occurs:

A new colormap is create
The color values for all entries are copied from *src-cmap*
All entries in *src-cmap* are freed.

If *src-cmap* was not created by the client with *alloc* **All**, then the pixels and planes allocated by the client using one of the following:

AllocColor
AllocColorCells
AllocColorPlanes
AllocNamedColor

¦ Copyright IBM Corp. 1985, 1991

5.13.23 CopyGC

src-gc, dst-gc: **GCONTEXT**
value-mask: **BITMASK**

Errors: **GContext, Value, Match, Alloc**

CopyGC copies components from *src-gc* (source) to *dst-gc* (destination). The source GC and the destination GC must have the same root and the same depth.

The *value-mask* specifies which components to copy. See **CreateGC** for a list of these components.

¦ Copyright IBM Corp. 1985, 1991

5.13.24 CopyPlane

src-drawable, dst-drawable: **DRAWABLE**
gc: **GCONTEXT**
src-x, src-y: **INT16**
width, height: **CARD16**
dst-x, dst-y: **INT16**
bit-plane: **CARD32**

Errors: **Drawable, GContext, Value, Match**

CopyPlane combines the *scr-drawable* (source) with the *dest-drawable* (destination). The source drawable and the destination drawable must have the same root, but the same depth.

The *bit-plane* must have exactly one bit set. Effectively, a pixmap of the depth as *dst-drawable* with the size specified by the source region is formed using the foreground and background pixels in *gc*.

The *bit-plane* in *src-drawable* contains a one bit for foreground.
The *bit-plane* in *src-drawable* contains a zero bit for background.

CopyPlane is the same as using the specified region of the source *bit-plane* as a stipple with a fill-style of **OpaqueStippled** for filling a rectangular area of the destination.

GC components are *function, plane-mask, foreground, background, subwindow-mode, graphics-exposures, clip-x-origin, clip-y-origin,* and *clip-mask*.

For more information on the exposure semantics, see **CopyArea** in this chapter.

¦ Copyright IBM Corp. 1985, 1991

5.13.25 CreateColormap

mid: **COLORMAP**
visual: **VISUALID**
window: **WINDOW**
alloc: {**None, All**}

Errors: **IDChoice, Window, Value, Match, Alloc**

CreateColormap creates a colormap of the specified *visual* type for the screen on which the *window* resides. It associates the identifier *mid* with type. The *visual* type must be supported by the screen.

The initial values of the colormap entries are undefined for classes **GrayScale**, **PseudoColor**, and **DirectColor**. For these classes, if *alloc* is **None**, clients can allocate the colormap entries.

The colormap entries for **StaticGray**, **StaticColor**, and **TrueColor** are defined values specific to the *visual*, but not defined by the Core protocol. The *alloc* variable for these classes must be specified as **None**.

If *alloc* is **All**, then the entire colormap is allocated writable. The initial value of allocated entries is undefined.

For **GrayScale** and **PseudoColor**, the effect is as thought **AllocColor** returned all pixel values from zero to *N*-1, where *N* is the colormap-entries value in the specified visual.

For **DirectColor**, the effect is as if an **AllocColorPlanes** returned a pixel value of zero and red-mask, green-mask, and blue-mask values containing the same bits as the corresponding masks in the specified visual.

However, in these cases, these entries cannot be freed with **FreeColors**.

¦ Copyright IBM Corp. 1985, 1991

5.13.26 CreateCursor

cid: **CURSOR**
source: **PIXMAP**
mask: **PIXMAP** or **None**
fore-red, fore-green, fore-blue: **CARD16**
back-red, back-green, back-blue: **CARD16**
x, y: **CARD16**

Errors: **IDChoice, Pixmap, Match, Alloc**

CreateCursor creates a cursor which is specified by the *cid* variable. The foreground and background RGB values must be specified, even if the server only has a **StaticGray** or **GrayScale** screen.

The foreground is used for the one bits in the source
The background is used for the zero bits in the source

The *source* pixmap and the *mask* pixmap must have depth of one and any size root. The *mask* pixmap does not need to be specified.

The *mask* pixmap defines the shape of the cursor, or the one bits in the *mask* define which *source* pixels will be displayed.

If the *mask* has zero bits, the corresponding bits of the *source* pixmap are ignored.
If no *mask* is given, all pixels of the *source* are displayed.

The *mask* and the *source* must be the same size.

The *x* and *y* coordinates define the hotspot. These coordinates are relative to the origin of the *source* and must be a point within the *source*.

The components of the cursor may be transformed arbitrarily to meet display limitations. The pixmaps can be freed immediately if no further explicit references are to be made to these pixmaps. Subsequent drawing in the source or the mask pixmap has an undefined effect on the cursor. The server may or may not make a copy of the pixmap.

¦ Copyright IBM Corp. 1985, 1991

5.13.27 CreateGC

cid: **GCONTEXT**
drawable: **DRAWABLE**
value-mask: **BITMASK**
value-list: **LISTofVALUE**

Errors: **IDChoice, Drawable, Pixmap, Font, Match, Value, Alloc**

CreateGC creates a graphics context. It assigns the identifier *cid* to it. The graphics context can be used with any destination drawable with the same root and depth as the specified *drawable*.

The *value-mask* and *value-list* specify which components are to be explicitly initialized. The graphics context components include:

Component	Type	Default
function	{**Clear, And, AndReverse, Copy, AndInverted, Noop, Xor, Or, Nor, Equiv, Invert, OrReverse, CopyInverted, OrInverted, Nand, Set**}	**Copy**
plane-mask	**CARD32**	all ones
foreground	**CARD32**	0
background	**CARD32**	1
line-width	**CARD16**	0
line-style	{**Solid, OnOffDash, DoubleDash**}	**Solid**
cap-style	{**NotLast, Butt, Round, Projecting**}	**Butt**
join-style	{**Miter, Round, Bevel**}	**Miter**
fill-style	{**Solid, Tiled, OpaqueStippled, Stippled**}	**Solid**
fill-rule	{**EvenOdd, Winding**}	
arc-mode	{**Chord, PieSlice**}	**PieSlice**
tile	**PIXMAP**	Pixmap of unspcified size filled with *foreground* pixel (client-specified pixel, if any, else zero). Subsequent changes to *foreground* do not affect this pixmap.
stipple	**PIXMAP**	Pixmap of unspecified size filled with ones.
tile-stipple-x-origin	**INT16**	0
tile-stipple-y-origin	**INT16**	0
font	**FONT**	<server-dependent-font>
subwindow-mode	{**ClipByChildren,**	**ClipByChildren**

¦ Copyright IBM Corp. 1985, 1991

	IncludeInferiors}	
graphics-exposures	**BOOL**	**True**
clip-x-origin	**INT16**	0
clip-y-origin	**INT16**	0
clip-mask	**PIXMAP** or **None**	**None**
dash-offset	**CARD16**	0
dashes	**CARD8**	4 (the first [4,4])

In graphics operations, given a source and destination pixel, the result is computed bitwise on corresponding bits of the pixels. That is, a Boolean operation is performed in each bit plane. The *plane-mask* restricts the operation to a subset of planes. The result is

((src FUNC dst) AND *plane-mask*) OR (dst AND (NOT *plane-mask*))

Range checking is not performed on the values for *foreground*, *background*, or *plane-mask*. These values are truncated to the appropriate number of bits. These values are defined as:

Function	**Operation**
Clear	0
And	src AND dst
AndReverse	src AND (NOT dst)
Copy	src
AndInverted	(NOT src) AND dst
NoOp	dst
Xor	src XOR dst
Or	src OR dst
Nor	(NOT src) AND (NOT dst)
Equiv	(NOT src) XOR dst
Invert	NOT dst
OrReverse	src OR (NOT dst)
CopyInverted	NOT src
OrInverted	(NOT src) OR dst
NAnd	(NOT src) OR (NOT dst)
Set	1

Line-width, measured in pixels, can be

a wide line (greater than or equal to one)

Wide lines are drawn centered on the path described by the graphics request. Unless otherwise specified by the *join-style* or *cap-style*, the bounding box of a wide line with endpoints [*x1, y1*], [*x2, y2*], and

¦ Copyright IBM Corp. 1985, 1991

width *w* 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)]
```

where *sn* is the sine of the angle of the line and *cs* is the cosine of the angle of the line. A pixel is part of the drawn line if the center of the pixel is fully inside the bounding box. If the center of the pixel is exactly on the bounding box, it is part of the line if and only if the interior is immediately to its right (with *x* increasing direction).

Pixels with centers on a horizontal edge are part of the line, if and only if, the interior is immediately below (with *y* increasing direction).

Real-point or fixed-point arithmetic is recommended for computing the corners of the line endpoints for lines greater than one pixel in width.

or a thin line (the special value zero)

Thin lines are drawn, one-pixel wide lines that use an unspecified, device-dependent algorithm. The two constraints on this algorithm are:

1. If a line is drawn unclipped from [*x1,y1*] to [*x2,y2*] and another line is drawn unclipped from [*x1+dx,y1+dy*] to [*x2+dx,y2+dy*], then a 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.

2. The effective set of points comprising a line cannot be affected by clipping. A point is touched in a clipped line if and only if the point lies inside the clipping region and the point would be touched by the line when drawn unclipped.

A wide line drawn from [*x1,y1*] to [*x2,y2*] always draws the same pixels as a wide line drawn from [*x2,y2*] to [*x1,y1*], not counting cap and join styles. It is encouraged that you make this property true for thin lines.

A *line-width* of zero differs from a *line-width* of one in which pixels are drawn. In general, drawing a thin line will be faster than drawing a wide line of width one, but thin lines may not mix well aesthetically with wide lines because of the different drawing algorithms. To obtain precise and uniform results across all displays, a client should always use a *line-width* of one, rather than a *line-width* of zero.

The *line-style* defines which sections of a line are drawn:

Line-Style	**Definition**
Solid	The full path of the line is drawn.
DoubleDash	The full path of the line is drawn, but the even dashes are filled differently than the odd dashes (see **Fill-Style**). **Butt** cap-style is used where even and odd dashes meet.
OnOffDash	Only the even dashes are drawn, and cap-style applies to all internal ends of the individual dashes (except **NotLast** is

¦ Copyright IBM Corp. 1985, 1991

treated as **Butt**).

The *cap-style* defines how the endpoints of a path are drawn:

Cap-Style	Definition
NotLast	This is the equivalent to **Butt**, except that for a *line-width* of zero or one the final endpoint is not drawn.
Butt	Square at the endpoint (perpendicular to the slope of the line), with no projection beyond.
Round	A circular arc with diameter equal to the *line-width*, centered on the endpoint; equivalent to **Butt** for *line-width* zero or one.
Projecting	Square at the end, but the path continues beyond the endpoint for a distance equal to half the line-width; equivalent to **Butt** for *line-width* zero or one.

The *join-style* defines how corners are drawn for wide lines:

Join-Style	Definition
Miter	The outer edges of the two lines extend to meet at an angle.
Round	A circular arc with diameter equal to the *line-width*, centered on the joinpoint.
Bevel	**Butt** endpoint styles, and then the triangular notch-filled.

For a line with coincident endpoints (*x1=x2, y1=y2*), when the *cap-style* is applied to both endpoints, the semantics depends on the *line-width* and the *cap-style*.

Cap and Line Combo	Definition
NotLast/thin	Device-dependent, but the desired effect is that nothing is drawn.
Butt/thin	Device-dependent, but the desired effect is that a single pixel is drawn.
Round/thin	Same as **Butt/thin**.
Projecting/thin	Same as **Butt/thin**.
Butt/wide	Nothing is drawn.
Round/wide	The closed path is a circle centered at the endpoint with diameter equal to the *line-width*.
Projecting/wide	The closed path is a square aligned with the coordinate axis centered at the endpoint, with sides equal to the *line-width*.

For a line with coincident endpoints (*x1=x2, y1=y2*), when the *join-style*

¦ Copyright IBM Corp. 1985, 1991

is applied at one or both endpoints, the effect is as if the line was 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 when the *cap-style* is applied at both endpoints.

The *tile-stipple* and clip origins are interpreted relative to the origin of whatever destination drawable is specified in a graphics request.

The *tile* pixmap must have the same root and depth as the **gcontext**. The *stipple* pixmap must have depth one and must have the same root as the **gcontext**. For *fill-style* **Stippled** (but not *fill-style* **OpaqueStippled**) the stipple pattern is tiled in a single plane and acts as an additional clip mask to be ANDed with the *clip-mask*. Any size pixmap can be used for tiling or stippling, although some sizes may be faster to use than others.

The *fill-style* defines the contents of the source for line, text, and fill requests. For all text and fill requests for line requests with *line-style* **Solid**, and for the even dashes for line requests with *line-style* **OnOffDash** or **DoubleDash**:

Fill-Style	**Definition**
Solid	Foreground.
Tiled	Tile.
OpaqueStippled	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.

Fill-Style	**Definition**
Stippled	Foreground masked by *stipple*.

For the odd dashes for line requests with *line-style* **DoubleDash**:

Fill-Style	**Definition**
Solid	Background.
Tiled	Same as for even dashes.
OpaqueStippled	Same as for even dashes.
Stippled	Background masked by stipple.

The allowed *dashes* value is actually a simplified form of the more general patterns that can be set with *SetDashes*. Specifying a value of *N* is equivalent to specifying the two-element list [*N, N*] in **SetDashes**. The value must be non-zero. The *dash-offset* and *dashes* are defined in the **SetDashes** request in this chapter.

The *clip-mask* restricts writes to the destination *drawable*. Only pixels where the *clip-mask* has a one bit are drawn; pixels are not drawn outside the area covered by the *clip-mask* or where the *clip-mask* has a zero bit. The *clip-mask* affects all graphics requests, but it does not clip sources. The *clip-mask* origin is interpreted relative to the origin of whatever destination drawable is specified in a graphics request. If a pixmap is specified as the *clip-mask*, it must have depth one and have the same root as the **gcontext**. If *clip-mask* is **None**, the pixels are drawn (regardless

¦ Copyright IBM Corp. 1985, 1991

of the clip origin). The *clip-mask* also can be set with the **SetClipRectangles** request.

For **ClipByChildren**, both source and destination windows are additionally clipped by all viewable **InputOutput** children. For **IncludeInferiors**, neither source nor destination window is clipped by inferiors; this will result in drawing through subwindow boundaries. **IncludeInferiors** can be used on windows with different depths, but the results are undefined by the Core protocol.

The *fill-rule* defines what pixels are inside and are drawn for paths given in **FillPoly** requests.

EvenOdd indicates that a point is inside if an infinite ray with the point as the origin crosses the path an odd number of times.

Winding indicates that a point is inside if an infinite ray with the point as an origin crosses an unequal number of clockwise and counterclockwise directed path segments.

A clockwise path is one which crosses the ray from left to right as observed from the point. A counter-clockwise segment is one which crosses the ray from right to left as observed from the point. The case where a directed line segment is coincident with the ray is uninteresting because you can choose a different ray which is not coincident with a segment.

For both fill rules, a point is infinitely small, and the path is an infinitely thin line. A pixel is inside 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).

The *arc-mode* controls filling in the **PolyFillArc** request.

The *graphics-exposures* flag controls the **GraphicsExposure** event generation for **CopyArea** and **CopyPlane** requests (and any similar requests defined by extensions).

The default component values are:

Component	**Default**
function	**Copy**
plane-mask	all ones
foreground	0
background	1
line-width	0
line-style	**Solid**
cap-style	**Butt**

¦ Copyright IBM Corp. 1985, 1991

join-style	**Miter**
fill-style	**Solid**
fill-rule	**EvenOdd**
arc-mode	**PieSlice**
tile	Pixmap of unspecified size filled with *foreground* pixel (client specified pixel if any, else zero). Subsequent changes to *foreground* do not affect this pixmap.
stipple	Pixmap of unspecified size filled with ones.
tile-stipple-x-origin	0
tile-stipple-y-origin	0
font	<server-dependent-font>
subwindow-mode	**ClipByChildren**
graphics-exposures	**True**
clip-x-origin	0
clip-y-origin	0
clip-mask	**None**
dash-offset	0
dashes	4 (the list [4, 4])

Storing a pixmap in a **gcontext** might or might not result in a copy being made. If the pixmap is later used as the destination for a graphics request, the change might or might not be reflected in the **gcontext**. If the pixmap is used simultaneously in a graphics request as both a destination and as a *tile* or *stipple*, the results are not defined.

It is quite likely that some amount of **gcontext** information will be cached in display hardware, and that such hardware can only cache a small number of **gcontexts**. Given the number and complexity of components, clients should view switching between **gcontexts** with nearly identical state as significantly more expensive than making minor changes to a single **gcontext**.

¦ Copyright IBM Corp. 1985, 1991

5.13.28 CreateGlyphCursor

cid: **CURSOR**
source-font: **FONT**
mask-font: **FONT** or **None**
source-char, mask-char: **CARD16**
fore-red, fore-green, fore-blue: **CARD16**
back-red, back-green, back-blue: **CARD16**

Errors: **IDChoice, Font, Value, Alloc**

CreateGlyphCursor creates a cursor and associates identifier *cid* with it. It obtains the the source and mask bitmaps from the specified font glyphs.

The *source-char* must be a defined glyph in *source-font* and *mask-char* must be a defined glyph in *mask-font*.

The *mask-font* and *mask-char* are optional.

The origins of the source and the defined mask glyphs are positione coincidently to define the hotspot. The source and mask do not need to have the same bounding box metrics. There are no restrictions on the placement of the hotspot relative to the bounding boxes. If no mask is given, all pixels of the source are displayed.

The *source-char* and *mask-char* are **CARD16**. For 2-byte matrix fonts, the 16 bit value is formed with *byte1* in the most-significant byte and *byte2* in the least-significant byte.

The cursor components can be transformed arbitrarily to meet display limitations. The fonts can be freed immediately if no further explicit references to them are made.

¦ Copyright IBM Corp. 1985, 1991

5.13.29 CreatePixmap

pid: **PIXMAP**
drawable: **DRAWABLE**
depth: **CARD8**
width, height: **CARD16**

Errors: **IDChoice, Drawable, Value, Alloc**

CreatePixmap creates a pixmap. It assigns the identifier *pid* to it.

The *width* and *height* fields must be non-zero.
The *depth* must be supported by the root of the specified *drawable*.
The initial contents of the pixmap are undefined

An **InputOnly** window can be used as a *drawable* in this request.

¦ Copyright IBM Corp. 1985, 1991

5.13.30 CreateWindow

window, parent: **WINDOW**
class: {**InputOutput, InputOnly, CopyFromParent**}
depth: **CARD8**
visual: **VISUALID** or **CopyFromParent**
x, y: **INT16**
width, height, borderwidth: **CARD16**
value-mask: **BITMASK**
value-list: **LISTofVALUE**

Errors: **IDChoice, Window, Pixmap, Colormap, Cursor, Match, Value, Alloc**

CreateWindow creates an unmapped window and assigns an identifier to it. Then, it generates a **CreateNotify** event.

If *class* is **CopyFromParent**, the class is taken from the parent.

If zero depth for class is **InputOutput** or **CopyFromParent**, the *depth* is taken from the parent.

If *visual* is **CopyFromParent**, the *visual* type is taken from the parent.

If *class* is **InputOutput**, the *visual* type and *depth* must be a combination supported for the screen. The *depth* does not need to be the same as the *parent*, but the *parent* must not be **InputOnly** class.

If the *class* is **InputOnly**, the *depth* must be zero and the *visual* type must be supported for the screen, but the *parent* can have any *depth* and *class*.

An **InputOnly** window cannot be used as a source or destination drawable for graphics requests. Otherwise, **InputOnly** and **InputOutput** windows act the same.

The window is placed on top in the stacking order with respect to siblings.

The *x* and *y* coordinates, which are relative to the origin of the parent, specify the position of the upper-left outer corner of the window.

The *width* and *height* specify the inside of the window, excluding the border. The *width* and *height* must be non-zero.

The *border-width* for an **InputOnly** window must be zero.

The *value-mask* and *value-list* specify window attributes that are to be explicitly initialized.

The following are possible values and the default values for the variables in **CreateWindow**.

Attribute	**Possible Values**	**Default Values**
background-pixmap	**PIXMAP**, **None**, **ParentRelative**	**None**
background-pixel	**CARD32**	No default
border-pixmap	**PIXMAP** or **CopyFromParent**	**CopyFromParent**
border-pixel	**CARD32**	No default

¦ Copyright IBM Corp. 1985, 1991

bit-gravity	**BITGRAVITY**	**Forget**
win-gravity	**WINGRAVITY**	**NorthWest**
backing-store	{**NotUseful, WhenMapped, Always**}	**NotUseful**
backing-planes	**CARD32**	all ones
backing-pixel	**CARD32**	**Zero**
save-under	**BOOL**	**False**
event-mask	**SETofEVENT**	**{}** (empty set)
do-not-propagate-mask	**SETofDEVICEEVENT**	**{}** (empty set)
override-redirect	**BOOL**	**False**
colormap	**COLORMAP** or **CopyFromParent**	**CopyFromParent**
cursor	**CURSOR** or **None**	**None**

The window attributes in **CreateWindow** include the following:

If *background-pixmap* is specified, it overrides the default background-pixmap. The *background-pixmap* and the window must have the same root and the same depth. Any size pixmap can be used, although some sizes may be faster than others.

- If *background-pixmap* is **None**, the window has no defined background.

- If *background-pixmap* is **ParentRelative**, the background of the parent is used, but the window must have the same depth as the parent.

- If the *background-pixmap* of the parent is **None**, the window will also have **None** for this variable. A copy of the parent *background-pixmap* is not made, but is re-examined each time the window background is required.

If *background-pixel* is specified, it overrides the default *background-pixmap* and any *background-pixmap* given explicitly. A pixmap of undefined size filled with background-pixel is used for the background.

- With *background-pixmap* of **ParentRelative**, the background tile origin aligns with the background of the parent tile origin. Otherwise, the background tile origin is always the window origin.

- When regions of the window are exposed and the server has not retained the contents, the server automatically tiles the regions with the background of the window unless the window has a *background-pixmap* of **None**.

- If the background is **None**, the previous screen contents are left in place if the window and its parent are the same depth. Otherwise, the initial contents of the exposed regions are undefined. Exposure events are then generated for the regions, even if the background is **None**.

For borders, the border tile origin is always the same as the background tile origin. Output to a window is always clipped to the inside of the window so that the border is never affected.

- If *border-pixmap* is given, it overrides the default *border-pixmap*. The border pixmap and the window must have the same root and the same depth. Any size pixmap can be used, although some sizes may be faster than others.

¦ Copyright IBM Corp. 1985, 1991

If *border-pixmap* is **CopyFromParent**, the border pixmap of the parent is copied. Subsequent changes to the parent do not affect the child.

If *border-pixel* is given, it overrides the default *border-pixmap* and any *border-pixmap* given explicitly. A pixmap of undefined size filled with border-pixel is used for the border.

The *bit-gravity* defines the region of the window that should be retained if the window is resized.

The *win-gravity* defines how the window should be repositioned if the parent is resized. See "ConfigureWindow" in topic 5.13.19.

The *backing-store* can be set to one of the following:

- **WhenMapped**, which advises the server that maintaining contents of obscured regions when the window is mapped would be beneficial.

- **Always**, which advises the server that maintaining contents even when the window is unmapped would be beneficial. Even if the window is larger than the parent, the server should maintain complete contents, not just the region within the parent boundaries. If the server maintains contents, exposure events will not be generated, but the server may stop maintaining contents at any time.

- **NotUseful**, which advises the server that maintaining contents is unnecessary, although a server may still choose to maintain contents while the window is mapped.

If *save-under* is **True**, it advises the server that when this window is mapped, saving the contents of the windows it obscures would be beneficial.

The *backing-planes* indicates (with one bit) which bit planes of the window hold dynamic data that must be preserved in *backing-stores* and during *save-unders*.

The *backing-pixel* specifies the value to use in planes not covered by *backing-planes*. The server can save only the specified bit planes in the *backing-store* or *save-under* and regenerate the remaining planes with the specified pixel value. Any bits beyond the specified depth of the window in these values are ignored.

The *event-mask* defines which events the client is interested in for this window or for some event types for inferiors of the window.

The *do-not-propagate-mask* defines which events should not be propagated to ancestor windows when no client has the event type selected in this window.

The *override-redirect* specifies whether map and configure requests on this window should override a **SubstructureRedirect** on the parent. This is typically done to inform the window manager not to tamper with the window.

The *colormap* specifies the colormap that best reflects the true colors of the window. The colormap must have the same visual type as the

¦ Copyright IBM Corp. 1985, 1991

window.

If **CopyFromParent** is specified, the colormap of the parent is copied, if the window has the same visual type as the parent and the *colormap* variable is not **None**. Subsequent changes to the parent do not affect the child.

If a *cursor* is specified, it will be used whenever the pointer is in the window. If **None** is specified, the cursor of the parent will be used, and any change in the cursor of the parent will cause an immediate change in the displayed cursor.

The background pixmap, border pixmap and the cursor can be freed immediately if no further explicit references to them are made.

Subsequent drawing into the background or border pixmap has an undefined effect on the window state; the server might or might not make a copy of the pixmap.

The only defined attributes for **InputOnly** windows are *win-gravity, event-mask, do-not-propagate-mask, override-redirect*, and *cursor*. Specifying other attributes for **InputOnly** windows generates an error.

¦ Copyright IBM Corp. 1985, 1991

5.13.31 DeleteProperty

window: **WINDOW**
property: **ATOM**

Errors: **Window, Atom**

DeleteProperty deletes the property from the specified window and generates a **PropertyNotify** event on the window if the property exists.

¦ Copyright IBM Corp. 1985, 1991

5.13.32 DestroySubwindows

window: **WINDOW**

Errors: **Window**

DestroySubwindows performs a **DestroyWindow** on all children of the window in a bottom-to-top stacking order.

¦ Copyright IBM Corp. 1985, 1991

5.13.33 DestroyWindow

window: **WINDOW**

Errors: **Window**

DestroyWindow destroys the window and all its inferior windows. (It has no effect on the root window.) If the *window* is mapped, an **UnmapWindow** request is performed. A **DestroyNotify** event is generated for each window.

When a **DestroyNotify** event is generated, it is generated on all inferior windows before being generated on the window. Otherwise, the ordering among siblings and across subhierarchies is not constrained.

Normal exposure processing on formerly obscured windows is performed.

¦ Copyright IBM Corp. 1985, 1991

5.13.34 FillPoly

drawable: **DRAWABLE**
gc: **GCONTEXT**
shape: {**Complex, Nonconvex, Convex**}
coordinate-mode: {**Origin, Previous**}
points: **LISTofPOINT**

Errors: **Drawable, GContext, Match, Value**

FillPoly fills the region defined by the specified set of points. The set of points is closed automatically if the last *point* in the list does not coincide with the first *point*. No pixel of the region is drawn more than once.

The first point is always relative to the origin of the *drawable*. The remaining points are relative to the origin of that point (**Origin**) or to the origin of the previous point (**Previous**) depending on the *coordinate-mode*.

The *shape* parameter may be used by the server to improve performance. If *shape* is:

- **Complex**, the path may self-intersect.

- **Nonconvex**, the path does not self-intersect, but the *shape* is not completely convex. If **Nonconvex** is specified for a self-intersecting path, the graphics results are undefined.

  If **Nonconvex** or **Complex** is known by the client, specifying **Nonconvex** over **Complex** may improve performance.

- **Convex**, the path is wholly convex.

If **Nonconvex** or **Complex** is known by the client, specifying **Convex** may improve performance.

If **Convex** is specified for a path that is not *convex*, the graphics results are undefined.

FillPoly uses the **GC** components *function, plane-mask, fill-style, fill-rule, subwindow-mode, clip-x-origin, clip-y-origin,* and *clip-mask*. It also uses the **GC** mode-dependent components *foreground, background, tile, stipple, tile-stipple-x-origin,* and *tile-stipple-y-origin*.

¦ Copyright IBM Corp. 1985, 1991

5.13.35 ForceScreenSaver

mode: {**Activate, Reset**}

Errors: **Value**

ForceScreenSaver activates the screen-saver if the *mode* is **Activate** and the screen-saver is currently deactivated, even if screen-saver has been disabled with a timeout value of zero.

If the mode is **Reset** and screen-saver is currently enabled, then screen-saver is deactivated (if it was activated) and the activation timer is reset to its initial state as if device input had just been received.

¦ Copyright IBM Corp. 1985, 1991

5.13.36 FreeColormap

cmap: **COLORMAP**

Errors: **Colormap**

FreeColormap deletes the association between the resource ID and the colormap. If the colormap is an installed colormap for a screen, it is uninstalled. (See "UninstallColormap" in topic 5.13.117.)

If the *cmap* is defined by **CreateWindow** or **ChangeWindowAttributes**, as the colormap for a window, the colormap for the window is changed to **None**, and a **ColormapNotify** event is generated.

If *cmap* is set to **None**, the colors, displayed for a window, are not defined by the protocol.

FreeColormap has no effect on a default colormap for a screen.

¦ Copyright IBM Corp. 1985, 1991

5.13.37 FreeColors

cmap: **COLORMAP**
pixel: **LISTofCARD32**
plane-mask: **CARD32**

Errors: **Colormap, Access, Value**

FreeColors frees all *pixels* allocated by the client with **AllocColor**, **AllocNamedColor**, **AllocColorCells**, and **AllocColorPlanes**.

The *plane-mask* should not have any bits in common with the *pixels*. The set of all *pixels* is produced by ORing together subsets of *plane-mask* with *pixels*.

Freeing an individual pixel obtained from **AllocColorPlanes** may not actually allow its reuse until all of its related *pixels* are also freed.

All *pixels* specified, allocated by the client in *cmap*, are freed even if one or more *pixels* produce an error.

A **Value** error is generated if a specified pixel is not a valid index into *cmap*. An **Access** error is generated if a specified pixel is not allocated by the client or is unallocated or is only allocated by another client.

¦ Copyright IBM Corp. 1985, 1991

5.13.38 FreeCursor

cursor: **CURSOR**

Errors: **Cursor**

FreeCursor deletes the association between the resource ID and the cursor. The cursor storage will be freed when no other resource references it.

¦ Copyright IBM Corp. 1985, 1991

5.13.39 FreeGC

gc: **GCONTEXT**

Errors: **GContext**

FreeGC deletes the association between the resource ID and the **gcontext** and destroys the **gcontext**.

¦ Copyright IBM Corp. 1985, 1991

5.13.40 FreePixmap

pixmap: **PIXMAP**

Errors: **Pixmap**

FreePixmap deletes the association between the resource ID and the pixmap. The pixmap storage will be freed when no other resource references it.

¦ Copyright IBM Corp. 1985, 1991

5.13.41 GetAtomName

```
   atom: ATOM
=>
   name: STRING8

      Errors: Atom
```

The **GetAtomName** returns the name for the given atom.

¦ Copyright IBM Corp. 1985, 1991

5.13.42 GetFontPath

```
=>
   path: LISTofSTRING8
```

GetFontPath returns the current search path for fonts.

¦ Copyright IBM Corp. 1985, 1991

5.13.43 GetGeometry

drawable: **DRAWABLE**
=>
root: **WINDOW**
depth: **CARD8**
x, y: **INT16**
width, height, border-width: **CARD16**

Errors: **Drawable**

GetGeometry returns the root and (current) geometry of the drawable.

The *depth* is the number of bits per pixel for the object.

The *x, y,* and *border-width* variables are always zero for pixmaps.

For a window, the *x* and *y* coordinates, which are relative to the origin of the parent, specify the upper-left outer corner of the window.

For a window, the *width* and *height* specify the inside of the window, excluding the border.

An **InputOnly** window can be used as a drawable to this request.

¦ Copyright IBM Corp. 1985, 1991

5.13.44 GetImage

drawable: **DRAWABLE**
x, y: **INT16**
width, height: **CARD16**
plane-mask: **CARD32**
format: {**XYPixmap, ZPixmap**}
=>
depth: **CARD8**
visual: **VISUALID** or **None**
data: **LISTofBYTE**

Errors: **Drawable, Value, Match**

GetImage returns the contents of the given rectangle of the specified drawable in the specified format. This request is intended for rudimentary hardcopy support.

The *x* and *y* coordinates, which are relative to the origin of the *drawable*, define the upper-left corner of the rectangle.

If *format* is **XYPixmap**, only the bit planes specified in *plane-mask* are transmitted with the planes appearing from most-signficant to least-significant in bit order.

If *format* is **ZPixmap**, the bits in all planes not specified in *plane-mask* are transmitted as zero.

Range checking is not performed on *plane-mask*. Therefore, extraneous bits are ignored.

The *depth* returned is the *depth* specified when the *drawable* was created. It is the depth specified in a **FORMAT** structure in the connection setup, not a bits-per-pixel component.

If the *drawable* is:

A window, its *visual* type is returned. It must be a a mapped window. This window must have no inferiors or overlapping windows. The specified rectangle of the window should be fully visible on the screen and completely contained within the outside edges of the window. The borders of the window can be included and read with this request.

A pixmap, the *visual* is **None**, and the given rectangle must be contained wholly within the pixmap.

If the window has a backing-store, then the backing-store contents returned are for regions of the window that are obscured by non-inferior windows. Otherwise, the contents returned of such obscured regions are undefined, and the contents returned of visible regions of inferior windows of depths different than the depth for the specified window are also undefined.

¦ Copyright IBM Corp. 1985, 1991

5.13.45 GetInputFocus

```
=>
   focus: WINDOW or PointerRoot or None
   revert-to: {Parent, PointerRoot, None}
```

GetInputFocus returns the current focus state.

¦ Copyright IBM Corp. 1985, 1991

5.13.46 GetKeyboardControl

=>

key-click-percent: **CARD8**
bell-percent: **CARD8**
bell-pitch: **CARD16**
bell-duration: **CARD16**
led-mask: **CARD32**
global-auto-repeat: {**On, Off**}
auto-repeats: **LISTofCARD8**

GetKeyboardControl returns the current control values for the keyboard.

For the LEDs, the least-significant bit of *led-mask* corresponds to LED one. Each one bit in *led-mask* indicates that an LED that is lit.

The *auto-repeats* value is a bit vector. Each one bit indicates that *auto-repeat* is enabled for the corresponding key. The vector is represented as 32 bytes.

Byte *N* (from zero) contains the bits for keys 8*N* to 8*N*+7, with the least-significant bit in the byte representing key 8*N*.

¦ Copyright IBM Corp. 1985, 1991

5.13.47 GetKeyboardMapping

```
    first-keycode: KEYCODE
    count: CARD8
=>
    keysyms-per-keycode: CARD8
    keysyms: LISTofKEYSYM

      Errors: Value
```

GetKeyboardMapping returns the symbols for the specified number of keycodes, starting with the specified keycode.

The *first-keycode* must be greater than or equal to *min-keycode* as it was returned in the connection setup. The result of

```
first-keycode + count -1
```

must be less than or equal to *max-keycode* as it was returned in the connection setup.

The number of elements in the *keysyms* list is

```
count * keysyms-per-keycode
```

and KEYSYM number *N* (counting from zero) for keycode *K* has an index (counting from zero) of

```
K - first-keycode) * keysyms-per-keycode + N
```

in *keysyms*.

The *keysyms-per-keycode* is chosen arbitrarily by the server to be large enough to report all requested symbols; a special KEYSYM value of **NoSymbol** is used to fill in unused elements for individual keycodes.

¦ Copyright IBM Corp. 1985, 1991

5.13.48 GetModifierMapping

=>

keycodes-per-modifier: **CARD8**
keycodes: **LISTofKEYCODE**

GetModifierMapping returns the *keycodes* of the keys being used as modifiers.

The number of *keycodes* in the list is **8*keycodes-per-modifier**. The *keycodes* are divided into eight sets, with each set containing *keycodes-per-modifier* elements. The sets are assigned in the modifiers in the following order: **Shift, Lock, Control, Mod1, Mod2, Mod3, Mod4, and Mod5**.

The *keycodes-per-modifier* is chosen arbitrarily by the server. Zeroes are used to fill in unused elements within each set. If only zero values are given in a set, the use of the corresponding modifier has been disabled. The order of *keycodes* within each set is chosen arbitrarily by the server.

¦ Copyright IBM Corp. 1985, 1991

5.13.49 GetMotionEvents

```
   start, stop: TIMESTAMP or CurrentTime
   window: WINDOW
=>
   events: LISTofTIMECOORD
        where
   TIMECOORD: {x, y: CARD16
                        time: TIMESTAMP}

      Errors: Window
```

GetMotionEvents returns all events in the motion history buffer that fall between the specified *start* and *stop* times (inclusive) and that have coordinates lying within (including borders) the specified *window* at its present placement.

If the *start* time is later than the *stop* time, or if the *start* time is in the future, no events are returned.

If the *stop* time is in the future, it is equivalent to specifying **CurrentTime**.

The *x* and *y* coordinates are reported relative to the origin of the window.

¦ Copyright IBM Corp. 1985, 1991

5.13.50 GetPointerControl

=>
    acceleration-numerator, acceleration-denominator: **CARD16**
    threshold: **CARD16**

GetPointerControl returns the current acceleration and *threshold* for the pointer.

¦ Copyright IBM Corp. 1985, 1991

5.13.51 GetPointerMapping

=>

map: **LISTofCARD8**

GetPointerMapping returns the current mapping of the pointer. Elements of the list are indexed starting from one. The length of the list indicates the number of physical buttons.

The nominal mapping for a pointer is the identity mapping: **map[i]=i**.

¦ Copyright IBM Corp. 1985, 1991

5.13.52 GetProperty

window: **WINDOW**
property: **ATOM**
type: **ATOM** or **AnyPropertyType**
long-offset, long-length: **CARD32**
delete: **BOOL**
=>
type: **ATOM** or **None**
format: {**0, 8, 16, 32**}
bytes-after: **CARD32**
value: **LISTofINT8** or **LISTofINT16** or **LISTofINT32**

Errors: **Window, Atom, Value**

GetProperty returns a specified property with the following provisions:

If the specified property does not exist for the specified window then

- The return *type* is **None**
- The *format* and *bytes-after* are zero
- The *value* is empty.

The *delete* argument is ignored.

If the specified property exists but its *type* does not match the specified type,

- The return *type* is the actual type of the property,
- The *format* is the actual format of the property (never zero),
- The *bytes-after* is the length of the property in bytes (even if the format is **16** or **32**)
- The *value* is empty.

The *delete* argument is ignored.

If the specified property exists, and either **AnyPropertyType** is specified or the specified type matches the actual type of the property, then

- The return *type* is the actual type of the property
- The *format* is the actual format of the property (never zero)
- The *bytes-after* and *value* are:

```
N = actual length of the stored property in bytes
    (even if the format is 16 or 32)
I = 4 * long-offset
T = N - I
L = MINIMUM(T, 4 * long-length)
A = N - (I + L).
```

The *value* returned starts at byte index **I** in the property (indexing from zero), and its property length in bytes is **L**.

If *long-offset* is given with **L** negative, a **Value** error occurs.

The value of *bytes-after* is **A**, giving the number of trailing unread

¦ Copyright IBM Corp. 1985, 1991

bytes in the stored property.

If *delete* is **True** and *bytes-after* is zero, the property is also deleted from the window and a **PropertyNotify** event is generated on the window.

¦ Copyright IBM Corp. 1985, 1991

5.13.53 GetScreenSaver

```
=>
    timeout, interval: CARD16
    prefer-blanking: {Yes, No}
    allow-exposures: {Yes, No}
```

GetScreenSaver returns the current screen-saver control values.

¦ Copyright IBM Corp. 1985, 1991

5.13.54 GetSelectionOwner

selection: **ATOM**
=>
owner: **WINDOW** or **None**

Errors: **Atom**

GetSelectionOwner returns the current owner window of the specified *selection*. If **None** is returned, there is no *owner* for the selection.

¦ Copyright IBM Corp. 1985, 1991

5.13.55 GetWindowAttributes

window: **WINDOW**
=>
visual: **VISUALID**
class: {**InputOutput, InputOnly**}
bit-gravity: **BITGRAVITY**
win-gravity: **WINGRAVITY**
backing store: {**NotUseful, WhenMapped, Always**}
backing-planes: **CARD32**
backing-pixel: **CARD32**
save-under: **BOOL**
colormap: **COLORMAP** or **None**
map-is-installed: **BOOL**
map-state: {**Unmapped, Unviewable, Viewable**}
all-event-masks, your-event-mask: **SETofEVENT**
do-not-propagate-mask: **SETofDEVICEEVENT**
override-redirect: **BOOL**

Errors: **Window**

GetWindowAttributes returns current attributes of the window.

The *map-state* is **Unviewable** if the window is mapped but an ancestor is unmapped.

The *all-event-masks* field is the inclusive-OR of all event masks selected on the window by clients.

The *your-event-mask* is the event mask selected by the querying client.

¦ Copyright IBM Corp. 1985, 1991

5.13.56 GrabButton

modifiers: **SETofKEYMASK** or **AnyModifier**
button: **BUTTON** or **AnyButton**
grab-window: **WINDOW**
owner-events: **BOOL**
event-mask: **SETofPOINTEREVENT**
pointer-mode, keyboard-mode: {**Synchronous, Asynchronous**}
confine-to: **WINDOW** or **None**
cursor: **CURSOR** or **None**

Errors: **Cursor, Window, Value, Access**

GrabButton establishes a passive grab. In the future, if the following occurs:

The pointer is not grabbe

The specified button is logically pressed when the specified modifie keys are logically down

The grab-window contains the pointe

The confine-to window (if any) is viewabl

A passive grab on the same button/key combination does not exist o any ancestor of grab-window,

Then,

The pointer is actively grabbed (see **GrabPointer**)
The last-pointer-grab *time* is set to the time at which the *button* was pressed (as transmitted in the **ButtonPress** event)
The **ButtonPress** event is reported.

The active grab is terminated automatically when all buttons are released (independent of the state of *modifier* keys).

A *modifier* of **AnyModifier** is equivalent to issuing the request for all possible *modifier* combinations (including the combination of no *modifiers*). It is not required that all *modifiers* specified have currently assigned keycodes.

A button of **AnyButton** is equivalent to issuing the request for all possible *buttons*. Otherwise, it is not required that the *button* specified currently be assigned to a physical *button*.

An **Access** error is generated if another client has already issued a **GrabButton** with the same button-key combination on the same window.

When using **AnyModifier** or **AnyButton**, the request fails completely or no grabs are established, and generates an **Access** error if there is a conflicting grab for any button/key combination. The request has no effect on an active grab.

See **GrabPointer** in this chapter for a discussion of the other variables.

¦ Copyright IBM Corp. 1985, 1991

5.13.57 GrabKey

key: **KEYCODE** or **AnyKey**
modifiers: **SETofKEYMASK** or **AnyModifier**
grab-window: **WINDOW**
owner-events: **BOOL**
pointer-mode, keyboard-mode: {**Synchronous, Asynchronous**}

Errors: **Window, Value, Access**

GrabKey establishes a passive grab on the keyboard. In the future, if the following occurs:

The keyboard is not grabbe

The specified *key*, which can be a modifier key, is logically pressed when the specified modifier keys are logically down

No other modifier keys are logically dow

The *grab-window* is an ancestor of the focus window, or is the focus window or is a descendent of the focus window and contains the pointer,

A passive grab on the same key combination does not exist on an ancestor of *grab-window*,

Then,

The keyboard is actively grabbed (see **GrabKeyboard**)

The last-keyboard-grab time is set to the time at which the key wa pressed (as transmitted by the **KeyPress** event)

The **KeyPress** event is reported.

The active grab is terminated automatically when the specified key has been released (independent of the state of the *modifier keys*).

A *modifier* of **AnyModifier** is equivalent to issuing the request for all possible modifier combinations (including the combination of no *modifiers*). It is not required that all *modifiers* specified have currently assigned keycodes.

A *key* of **AnyKey** is equivalent to issuing the request for all possible keycodes. Otherwise, the key must be in the range specified by min-keycode and max-keycode in the connection setup or else a **Value** error occurs.

An **Access** error is generated if another client has issued a **GrabKey** with the same key combination on the same window. When using **AnyModifier** or **AnyKey**, the request fails completely (no grabs are established) and an **Access** error is generated if there is a conflicting grab for any combination.

See **GrabKeyboard** in this chapter for a discussion about *pointer-mode, keyboard-mode,* and *owner-events* arguments.

¦ Copyright IBM Corp. 1985, 1991

5.13.58 GrabKeyboard

grab-window: **WINDOW**
owner-events: **BOOL**
pointer-mode, keyboard-mode: {**Synchronous, Asynchronous**}
time: **TIMESTAMP** or **CurrentTime**
=>
status: {**Success, AlreadyGrabbed, Frozen, InvalidTime, NotViewable**}

Errors: **Window, Value**

GrabKeyboard actively grabs control of the keyboard. Further key events are reported only to the grabbing client. This request overrides any active keyboard grab by this client.

If *owner-events* is:

False, all generated key events are reported with respect to *grab-window*.

True, and a generated key event is normally reported to this client, the event is still reported. If the key event normally is not reported to this client, it will be reported with respect to the *grab-window*.

Both **KeyPress** and **KeyRelease** events are always reported, independent of any event selection made by the client.

If *pointer-mode* is:

Asynchronous, pointer event processing is unaffected by activation of the grab.

Synchronous, the pointer appears to the protocol to freeze, and no further pointer events are generated by the server until the grabbing client issues a releasing **AllowEvents** request. Keyboard changes made while the keyboard is frozen are queued for later processing.

If *keyboard-mode* is:

Asynchronous, keyboard event processing continues normally. If it is **Asynchronous** and the keyboard is currently frozen by this client, then keyboard event processing is resumed.

Synchronous, the keyboard appears to the protocol to freeze and no further keyboard events are generated by the server until the grabbing client issues a releasing **AllowEvents** request. Keyboard changes made while the keyboard is frozen are queued for later processing.

The request fails if *status* is:

AlreadyGrabbed and the keyboard is actively grabbed by some other client

Frozen and the keyboard is frozen by an active grab of another client

NotViewable and *grab-window* is not viewable

InvalidTime and the specified time is earlier than the last-keyboard-grab *time* or later than the current server *time*.

¦ Copyright IBM Corp. 1985, 1991

(Otherwise, the last-keyboard-grab *time* is set to the specified *time* and **CurrentTime** replaces the current server *time*.)

This request generates **FocusIn** and **FocusOut** events.

¦ Copyright IBM Corp. 1985, 1991

5.13.59 GrabPointer

grab-window: **WINDOW**
owner-events: **BOOL**
event-mask: **SETofPOINTEREVENT**
pointer-mode, keyboard-mode: {**Synchronous, Asynchronous**}
confine-to: **WINDOW** or **None**
cursor: **CURSOR** or **None**
time: **TIMESTAMP** or **CurrentTime**
=>
status: {**Success, AlreadyGrabbed, Frozen, InvalidTime, NotViewable**}

Errors: **Cursor, Window, Value**

GrabPointer actively grabs control of the pointer. Further pointer events are only reported to the grabbing client. The request overrides any active pointer grab by this client.

If *owner-events* is:

False, all generated key events also selected by *event-mask* are reported with respect to *grab-window*. Unreported events are discarded.

True and a generated key event that is also selected by *event-mask* is normally reported to this client, the report is made. If the key event is not reported normally, it is reported with respect to the *grab-window*. Unreported events are discarded.

If *pointer-mode* is:

Asynchronous, pointer event processing is unaffected by activation of the grab and event processing continues normally. If the pointer is currently frozen by this client, processing of pointer events is resumed.

Synchronous, the pointer appears to the protocol to freeze, and no further pointer events are generated by the server until the grabbing client issues a releasing **AllowEvents** request. Keyboard changes made while the keyboard is frozen are queued for later processing.

If *keyboard-mode* is:

Asynchronous, keyboard event processing continues normally.

Synchronous, the keyboard appears to the protocol to freeze and no further keyboard events are generated by the server until the grabbing client issues a releasing **AllowEvents** request. Keyboard changes made while the keyboard is frozen are queued for later processing.

If a *cursor* is specified, it is displayed regardless of where the pointer is located. If a *cursor* is not specified, then, when the pointer is in the *grab-window* or one of its subwindows, the normal *cursor* for that window is displayed. Otherwise, the *cursor* for the *grab-window* is displayed.

If a *confine-to* window is specified, then the pointer is restricted to that window. The *confine-to* window does not need to have a relationship to the *grab-window*.

¦ Copyright IBM Corp. 1985, 1991

If the pointer is not in the *confine-to* window initially, it is warped automatically to the closest edge (and enter or leave events are generated normally) just before the grab activates. If the *confine-to* window is reconfigured subsequently, the pointer is warped automatically to keep it in the window.

GrabPointer fails if *status* is:

AlreadyGrabbed which indicates that the keyboard is actively grabbed by some other client.

Frozen which indicates that the keyboard is frozen by an active grab of another client.

NotViewable which indicates that the *grab-window* or *confine-to* window is not viewable.

InvalidTime which indicates that the specified time is earlier than the last-keyboard-grab time or later than the current server time. (Otherwise, the last-keyboard-grab time is set to the specified time and **CurrentTime** replaces the current server *time*.)

GrabPointer generates **EnterNotify** and **LeaveNotify** events.

¦ Copyright IBM Corp. 1985, 1991

5.13.60 GrabServer

GrabServer disables processing of requests and close-downs on all connections, other than the one this request arrived on when initiated.

¦ Copyright IBM Corp. 1985, 1991

5.13.61 ImageText16

drawable: **DRAWABLE**
gc: **GCONTEXT**
x, y: **INT16**
string: **STRING16**

Errors: **Drawable, GContext, Match**

ImageText16 fills in a destination rectangle with the background pixel defined in *gc* and paints the text with the foreground pixel. It uses 2-byte (or 16-bit) characters. For fonts defined with linear indexing rather than 2-byte matrix indexing, the server interprets each **CHAR2B** as a 16-bit number that has been transmitted with the *byte1* of the **CHAR2B** as the most-significant byte.

¦ Copyright IBM Corp. 1985, 1991

5.13.62 ImageText8

drawable: **DRAWABLE**
gc: **GCONTEXT**
x, y: **INT16**
string: **STRING8**

Errors: **Drawable, GContext, Match**

ImageText8 fills in a destination rectangle with the background pixel defined in *gc* and paints the text with the foreground pixel.

The *x* and *y* coordinates, which are relative to the origin of the *drawable*, specify the baseline starting position of the origin of the initial character.

The upper-left corner of the filled rectangle is at

```
[x, y - font-ascent]
```

the width is overall-width, and the height is

```
font-ascent + font-descent
```

where *overall-width*, *font-ascent*, and *font-descent* are the same as returned by **QueryTextExtents** using *gc* and *string*.

The function and fill-style defined in *gc* are ignored for this request; the effective function is **Copy** and the effective fill-style **Solid**.

For fonts defined with 2-byte matrix indexing, each **STRING8** byte is interpreted as a *byte2* value of a **CHAR2B** with a *byte1* value of zero.

ImageText8 uses the **GC** components *plane-mask, foreground, background, font, subwindow-mode, clip-x-origin, clip-y-origin,* and *clip-mask*.

¦ Copyright IBM Corp. 1985, 1991

5.13.63 InstallColormap

cmap: **COLORMAP**

Errors: **Colormap**

InstallColormap installs this colormap for the screen. All windows associated with this colormap are displayed immediately with true colors. Depending on the server, this request can install or uninstall additional colormaps. Regardless of the server, the required list must remain installed.

A subset of the installed colormaps viewed as an ordered list is the required list. The length of the required list is at most *M*, where *M* is the minimum installed colormaps specified for the screen in the connection setup. The required list is maintained as follows:

When a colormap is an explicit argument to **InstallColormap**, it is added to the top of the required list, and the required list is truncated at the end, if necessary, to keep the length of the list at *M*.

When a colormap is an explicit argument to **UninstallColormap** and it is in the required list, it is removed from the list.

A colormap is not added to the required list implicitly by the server when it is installed, nor does the server uninstall a colormap explicitly from the required list.

If the specified colormap is not an installed colormap, a **ColormapNotify** event is generated on every window having *cmap* as an attribute. In addition, for every other colormap that is installed or uninstalled as a result of this request, a **ColormapNotify** event is generated on every window having that colormap as an attribute.

Initially, the default colormap for a screen is installed but is not in the required list.

¦ Copyright IBM Corp. 1985, 1991

5.13.64 InternAtom

name: **STRING8**
only-if-exists: **BOOL**
=>
atom: **ATOM** or **None**

Errors: **Value, Alloc**

InternAtom returns the atom for the given name.

If *only-if-exists* is **False**, the atom is created if it does not exist. The atom is a case-sensitive string.

The lifetime of an atom is not tied to the interning client. Atoms remain defined until server reset. For more information, see "Closing Connections to the Server" in topic 5.10.

¦ Copyright IBM Corp. 1985, 1991

5.13.65 KillClient

resource: **CARD32** or **AllTemporary**

Errors: **Value**

KillClient forces a close-down of the client that created the resource for a valid *resource*. If the client is terminated already in **RetainPermanent** or **RetainTemporary** mode, all client resources are destroyed. For more information, see "Closing Connections to the Server" in topic 5.10.

If **AllTemporary** is specified, the *resources* of all terminated clients in **RetainTemporary** mode are destroyed.

¦ Copyright IBM Corp. 1985, 1991

5.13.66 ListExtensions

=>
    names: **LISTofSTRING8**

ListExtensions returns a list of all extensions supported by the server.

¦ Copyright IBM Corp. 1985, 1991

5.13.67 ListFonts

```
   pattern: STRING8
   max-names: CARD16
=>
   names: LISTofSTRING8
```

ListFonts returns a list of fonts that match the specified pattern. The *pattern*, which is not case-sensitive, should use the ISO Latin-1 encoding. In the *pattern*, the "?" character (octal value 77) will match any single character, and the character "*" (octal value 52) will match any number of characters.

The *names* returned are lowercase.

¦ Copyright IBM Corp. 1985, 1991

5.13.68 ListFontsWithInfo

```
   pattern: STRING8
   max-names: CARD16
=>+
   name: STRING8
   info: FONTINFO
   replies-hint: CARD32

    where

        FONTINFO: <same type definition as in QueryFont>
```

ListFontsWithInfo returns a list with information about each font. This list is the same as the list returned by **QueryFont**, except that the per-character metrics are not returned. **ListFontsWithInfo** can generate multiple replies. With each reply, *replies-hint* indicates how many more fonts will be returned. This number is a hint only. The number of fonts returned can be larger or smaller than the number in *replies-hint*. A zero value does not guarantee that no more fonts will be returned. After the font replies, a reply with a zero-length name is sent to indicate the end of the reply sequence. See "QueryFont" in topic 5.13.90.

¦ Copyright IBM Corp. 1985, 1991

5.13.69 ListHosts

=>
    mode: {**Enabled, Disabled**}
    hosts: **LISTofHOST**

ListHosts returns the hosts on the access control list. It also indicates whether use of the list at connection setup is currently enabled or disabled.

Each **HOST** is padded to a multiple of 4 bytes.

¦ Copyright IBM Corp. 1985, 1991

5.13.70 ListInstalledColormaps

window: **WINDOW**
=>
cmaps: **LISTofCOLORMAP**

Errors: **Window**

ListInstalledColormaps returns a list of the colormaps installed currently for the screen of the specified *window*. The order of colormaps is insignificant and there is no explicit indication of the required list. See "InstallColormap" in topic 5.13.63.

¦ Copyright IBM Corp. 1985, 1991

5.13.71 ListProperties

window: **WINDOW**
=>
atoms: **LISTofATOM**

Errors: **Window**

ListProperties returns the *atoms* of properties currently defined on the *window*.

¦ Copyright IBM Corp. 1985, 1991

5.13.72 LookupColor

```
    cmap: COLORMAP
    name: STRING8
=>
    exact-red, exact-green, exact-blue: CARD16
    visual-red, visual-green, visual-blue: CARD16

      Errors: Colormap, Name
```

LookupColor searches for the *string name* of a color with respect to the screen associated with *cmap*. It returns both the exact color values and the closest values provided by the hardware with respect to the visual type of *cmap*. The *name*, which is case-sensitive, should use the ISO Latin-1 encoding.

¦ Copyright IBM Corp. 1985, 1991

5.13.73 MapSubwindows

window: **WINDOW**

Errors: **Window**

MapSubwindows performs a **MapWindow** request on all unmapped children of the *window* in a top-to-bottom stacking order.

¦ Copyright IBM Corp. 1985, 1991

5.13.74 MapWindow

window: **WINDOW**

Errors: **Window**

MapWindow maps an unmapped window. If the override-redirect attribute of the *window* is **False** and another client has selected **SubstructureRedirect** on the parent, then a **MapRequest** event is generated, but the window remains unmapped. Otherwise, the window is mapped and a **MapNotify** event is generated.

If the window is now viewable and its contents have been discarded, then the window is tiled with its background and one or more exposure events are generated. If no background is defined, the existing screen contents are not altered. If a backing store is maintained while the window is unmapped, exposure events are not generated. If a backing store is maintained, a full-window exposure is always generated. Otherwise, only visible regions can be reported. Similar tiling and exposure takes place for any newly viewable inferiors.

This request has no effect if the window is already mapped.

¦ Copyright IBM Corp. 1985, 1991

5.13.75 NoOperation

This request has no arguments and no results, but the request length field can be non-zero, allowing the request to be any multiple of 4 bytes in length. The bytes contained in the request are uninterpreted by the server.

NoOperation can be used in its minimum 4-byte form as padding where necessary by client libraries that find it convenient to force requests to begin on 64-bit boundaries.

¦ Copyright IBM Corp. 1985, 1991

5.13.76 OpenFont

fid: **FONT**
name: **STRING8**

Errors: **IDChoice, Name, Alloc**

OpenFont loads the specified font, if necessary, and associates identifier *fid* with it. The font name is not case-sensitive.

Fonts are not associated with a particular screen and can be stored as a component of any graphics context.

¦ Copyright IBM Corp. 1985, 1991

5.13.77 PolyArc

drawable: **DRAWABLE**
gc: **GCONTEXT**
arcs: **LISTofARC**

Errors: **Drawable, GContext, Match**

PolyArc draws circular or elliptical arcs. Each *arc* is specified by a rectangle and two angles. The angles are signed integers in degrees scaled by 64. A positive sign indicates counterclockwise motion and a negative sign indicates clockwise motion. The start of the *arc* is specified by *angle1* relative to the three-o-clock position from the center of the rectangle. The path and extent of the *arc* is specified by *angle2* relative to the start of the *arc*. If the magnitude of *angle2* is greater than 360 degrees, it is truncated to 360 degrees. The *x* and *y* coordinates of the rectangle are relative to the origin of the *drawable*.

For an *arc* specified as [*x,y,w,h,a1,a2*], the origin of the major and minor axes is at [*x*+(*w*/2),*y*+(*h*/2)], and the infinitely thin path describing the entire circle/ellipse intersects the horizontal axis at [*x,y*+(*h*/2)] and [*x+w,y*+(*h*/2)] and intersects the vertical axis at [*x*+(*w*/2),*y*] and [*x*+(*w*/2),*y+h*]. These coordinates can be fractional; they are not truncated to discrete coordinates. The path should be defined by the ideal mathematical path. For a wide line with line-width *lw*, the bounding outlines for filling are given by the infinitely thin paths describing the *arcs*:

```
[x+dx/2, y+dy/2, w-dx,h-dy, a1, a2]
```

and

```
[x-lw/2, y-lw/2, w+lw, h+lw, a1, a2]
```

where

```
dx = min(lw,w)
dy = min(lw,h)
```

For an *arc* specified as [*x,y,w,h,a1,a2*], the angles must be specified in the effectively skewed coordinate system of the ellipse (for a circle, the angles and coordinate systems are identical). The relationship between these angles and angles expressed in the normal coordinate system of the screen (as measured with a protractor) is as follows:

skewed-angle = atan(tan(*normal-angle*) * *w/h*) + *adjust*

where *skewed-angle* and *normal-angle* are expressed in radians (rather than in degrees scaled by 64) in the range [0,2*PI], and where *atan* returns a value in the range [-PI/2,PI/2], where *adjust* is

0 for *normal-angle* in the range [0,PI/2]

PI for *normal-angle* in the range [PI/2,(3*PI)/2]

2*PI or *normal-angle* in the range [(3*PI)/2,2*PI]

The *arcs* are drawn in the order listed. If the last point in one *arc* coincides with the first point in the following *arc*, the two *arcs* will join correctly. If the first point in the first *arc* coincides with the

¦ Copyright IBM Corp. 1985, 1991

last point in the last *arc*, the two *arcs* will join correctly. For any given *arc*, no pixel is drawn more than once. If two *arcs* join correctly, the line-width is greater than zero, and the *arcs* intersect, no pixel is drawn more than once. Otherwise, the intersecting pixels of intersecting *arcs* are drawn several times. Specifying an *arc* with one endpoint and a clockwise extent draws the same pixels as specifying the other endpoint and an equivalent counterclockwise extent, except as it affects joins.

By specifying one axis to be zero, a horizontal or vertical line can be drawn.

Angles are computed based solely on the coordinate system, ignoring the aspect ratio.

PolyArc uses the **GC** components *function, plane-mask, line-width, line-style, cap-style, join-style, fill-style, subwindow-mode, clip-x-origin, clip-y-origin,* and *clip-mask*. It also uses the **GC** mode-dependent components *foreground, background, tile, stipple, tile-stipple-x-origin, tile-stipple-y-origin, dash-offset,* and *dashes*.

¦ Copyright IBM Corp. 1985, 1991

5.13.78 PolyFillArc

drawable: **DRAWABLE**
gc: **GCONTEXT**
arcs: **LISTofARC**

Errors: **Drawable, GContext, Match**

For each arc, **PolyFillArc** fills the region closed by the infinitely thin path described by the specified arc and one or two line segments, depending on the arc mode.

For **Chord**, the single line segment joining the endpoints of the arc is used. For **PieSlice**, the two line segments joining the endpoints of the arc with the center point are used. The *arcs* are as specified in the **PolyArc** request.

The *arcs* are filled in the order listed. For any given arc, no pixel is drawn more than once. If regions intersect, the intersecting pixels are drawn several times.

PolyFillArc uses the **GC** components *function, plane-mask, fill-style, arc-mode, subwindow-mode, clip-x-origin, clip-y-origin,* and *clip-mask*. It also uses the **GC** mode-dependent components *foreground, background, tile, stipple, tile-stipple-x-origin,* and *tile-stipple-y-origin*.

¦ Copyright IBM Corp. 1985, 1991

5.13.79 PolyFillRectangle

drawable: **DRAWABLE**
gc: **GCONTEXT**
rectangles: **LISTofRECTANGLE**

Errors: **Drawable, GContext, Match**

PolyFillRectangle fills the specified *rectangles* as if a four-point **FillPoly** was specified for each rectangle:

```
[x,y] [x+width,y] [x+width,y+height] [x,y+height]
```

The *x* and *y* coordinates of each *rectangle*, which are relative to the origin of the drawable, define the upper-left corner of the rectangle.

The *rectangles* are drawn in the order listed. For any rectangle, no pixel is drawn more than once. If the rectangles intersect, the intersecting pixels are drawn multiple times.

PolyFillRectangle uses the **GC** components *function, plane-mask, fill-style, subwindow-mode, clip-x-origin, clip-y-origin,* and *clip-mask*. It also uses the **GC** mode-dependent components *foreground, background, tile, stipple, tile-stipple-x-origin,* and *tile-stipple-y-origin*.

¦ Copyright IBM Corp. 1985, 1991

5.13.80 PolyPoint

drawable: **DRAWABLE**
gc: **GCONTEXT**
coordinate-mode: {**Origin, Previous**}
points: **LISTofPOINT**

Errors: **Drawable, GContext, Value, Match**

PolyPoint combines the foreground pixel in *gc* with the pixel at each point in the *drawable*. The *points* are drawn in the order listed.

The first point is always relative to the origin of the *drawable*. The other points are relative either to the origin of the drawable (**Origin**) or to the origin of the previous point (**Previous**) depending on the *coordinate-mode*.

PolyPoint uses the **GC** components *function, plane-mask, foreground, subwindow-mode, clip-x-origin, clip-y-origin,* and *clip-mask*.

¦ Copyright IBM Corp. 1985, 1991

5.13.81 PolyLine

drawable: **DRAWABLE**
gc: **GCONTEXT**
coordinate-mode: {**Origin, Previous**}
points: **LISTofPOINT**

Errors: **Drawable, GContext, Value, Match**

PolyLine draws lines between each pair of *points* (**point[i], point[i+1]**). The lines are drawn in the order listed. The lines join correctly at all intermediate points and, if the first and last points coincide, the first and last lines also join correctly.

For any given line, no pixel is drawn more than once. If thin lines intersect, the intersecting pixels are drawn multiple times. If wide lines intersect, the intersecting pixels are drawn only once, as though the entire **PolyLine** were a single, filled shape.

The first point is always relative to the origin of the *drawable*. The other points are relative to the origin of the drawable (**Origin**) or to the origin of the previous point (**Previous**) depending on the *coordinate-mode*.

PolyLine uses the **GC** components *function, plane-mask, line-width, line-style, cap-style, join-style, fill-style, subwindow-mode, clip-x-origin, clip-y-origin,* and *clip-mask*. It also uses the **GC** mode-dependent components *foreground, background, tile, stipple, tile-stipple-x-origin, tile-stipple-y-origin, dash-offset,* and *dashes*.

¦ Copyright IBM Corp. 1985, 1991

5.13.82 PolyRectangle

drawable: **DRAWABLE**
gc: **GCONTEXT**
rectangles: **LISTofRECTANGLE**

Errors: **Drawable, GContext, Match**

PolyRectangle draws the outlines of the specified *rectangles*, as if a five-point **PolyLine** was specified for each *rectangle*:

[x, y] [x + width, y + height] [x + width, y[x,y + height] [x, y]

The *x* and *y* coordinates of each *rectangle*, which are relative to the origin of the *drawable*, define the upper-left corner of the *rectangle*.

The *rectangles* are drawn in the order listed. For any *rectangle*, no pixel is drawn more than once. If *rectangles* intersect, the intersecting pixels are drawn several times.

PolyRectangle uses the **GC** components *function, plane-mask, line-width, line-style, join-style, fill-style, subwindow-mode, clip-x-origin, clip-y-origin,* and *clip-mask*. It also uses the **GC** mode-dependent components *foreground, background, tile, stipple, tile-stipple-x-origin, tile-stipple-y-origin, dash-offset,* and *dashes*.

¦ Copyright IBM Corp. 1985, 1991

5.13.83 PolySegment

drawable: **DRAWABLE**
gc: **GCONTEXT**
segments: **LISTofSEGMENT**

where
SEGMENT: [*x1, y1, x2, y2*: **INT16**]

Errors: **Drawable, GContext, Match**

PolySegment draws a line between [*x1, y1*] and [*x2, y2*] for each segment. The lines are drawn in the order listed. No joining is performed at coincident endpoints. For any given line, no pixel is drawn more than once. If lines intersect, the intersecting pixels are drawn multiple times.

PolySegment uses the **GC** components *function, plane-mask, line-width, line-style, cap-style, fill-style, subwindow-mode, clip-x-origin, clip-y-origin,* and *clip-mask*. It also uses the **GC** mode-dependent components *foreground, background, tile, stipple, tile-stipple-x-origin, tile-stipple-y-origin, dash-offset,* and *dashes*.

¦ Copyright IBM Corp. 1985, 1991

5.13.84 PolyText16

drawable: **DRAWABLE**
gc: **GCONTEXT**
x, y: **INT16**
items: **LISTofTEXTITEM16**

where
TEXTITEM16: **TEXTELT16** or **FONT**
TEXTELT16: [*delta*: **INT8**
string: **STRING16**]

Errors: **Drawable, GContext, Match, Font**

PolyText16 enables a font to be stored and used for subsequent text. This request is the same as **PolyText8** except that 2-byte (or 16-bit) characters are used.

For fonts defined with linear indexing rather than 2-byte matrix indexing, the server interprets each **CHAR2B** as a 16-bit number transmitted with the most-significant byte first (*byte1*).

¦ Copyright IBM Corp. 1985, 1991

5.13.85 PolyText8

drawable: **DRAWABLE**
gc: **GCONTEXT**
x, y: **INT16**
items: **LISTofTEXTITEM8**

where
TEXTITEM8: TEXTELT8 or **FONT**
TEXTELT8: [*delta*: **INT8**
string: **STRING8**]

Errors: **Drawable, GContext, Match, Font**

PolyText8 enables a font to be stored and used for subsequent text. Each text item is processed in turn. A font item causes the font to be stored in *gc*, and to be used for subsequent text; switching among fonts does not affect the next character origin.

The *x* and *y* coordinates, which are relative to the origin of the *drawable*, specify the baseline starting position (the initial character origin).

A text element *delta* specifies an additional change in the position along the X-axis before the string is drawn; the *delta* is always added to the character origin. Each character image, as defined by the font in *gc*, is treated as an additional mask for a fill operation on the *drawable*.

All contained **FONT**s are transmitted with the most-significant byte first.

If a **Font** error is generated for an item, the previous items may have been drawn.

For fonts defined with 2-byte matrix indexing, each **STRING8** byte is interpreted as a *byte2* value of a **CHAR2B** with a *byte1* value of zero.

PolyText8 uses the **GC** components *function, plane-mask, fill-style, font, subwindow-mode, clip-x-origin, clip-y-origin,* and *clip-mask*. It also uses the **GC** mode-dependent components *foreground, background, tile, stipple, tile-stipple-x-origin,* and *tile-stipple-y-origin*.

¦ Copyright IBM Corp. 1985, 1991

5.13.86 PutImage

drawable: **DRAWABLE**
gc: **GCONTEXT**
depth: **CARD8**
width, height: **CARD16**
dst-x, dst-y: **INT16**
left-pad: **CARD8**
format: {**Bitmap, XYPixmap, ZPixmap**}
data: **LISTofBYTE**

Errors: **Drawable, GContext, Match, Value**

PutImage combines an image with a rectangle of the specified *drawable*. The *dst-x* and *dst-y* coordinates are relative to origin of the destination *drawable*.

If **Bitmap** *format* is used, the depth must be one and the image must be in **XYFormat**.

The foreground pixel in *gc* defines the source for one bits in the image, and the background pixel defines the source for the zero bits.

For **XYPixmap** and **ZPixmap**, the depth must match the depth of the *drawable*.

For **XYPixmap**, the image must be in **XYFormat**.

For **ZPixmap**, the image must be in **ZFormat** for the specified depth.

The first *left-pad* bits in each scanline are ignored by the server; the image actually begins that number of bits into the data.

For **ZPixmap** format, the *left-pad* must be zero.

For **Bitmap** and **XYPixmap** format, the *left-pad* must be less than the *bitmap-scanline-pad* as specified in the server connection setup.

The *width* defines the width of the actual image and does not include *left-pad*.

PutImage uses the **GC** components *function, plane-mask, subwindow-mode, clip-x-origin, clip-y-origin,* and *clip-mask*. It also uses the **GC** mode-dependent components *foreground* and *background*.

¦ Copyright IBM Corp. 1985, 1991

5.13.87 QueryBestSize

```
    class: {Cursor, Tile, Stipple}
    drawable: DRAWABLE
    width, height: CARD16
=>
    width, height: CARD16

      Errors: Drawable, Value, Match
```

QueryBestSize returns the size closest to the argument size.

If *class* is **Cursor**, this is the largest cursor that can be displayed fully on the *drawable* (screen).

If *class* is **Tile**, this is the size that can be tiled fastest on the *drawable* which can be the screen, or the window class and depth. An **InputOnly** window cannot be used as the *drawable* for **Tile**.

If *class* is **Stipple**, this is the size that can be stippled fastest on the specified *drawable* which can be the screen, or the window class and depth. An **InputOnly** window cannot be used as the *drawable* for **Stipple**.

¦ Copyright IBM Corp. 1985, 1991

5.13.88 QueryColors

```
    cmap: COLORMAP
    pixel: LISTofCARD32
=>
    colors: LISTofRGB

    where
        RGB: [red, green, blue: CARD16]

      Errors: Colormap, Value
```

QueryColors returns the color values stored in *cmap* for the specified *pixels*. Values returned for an unallocated entry are undefined. A **Value** error is generated if a pixel is not a valid index into *cmap*.

¦ Copyright IBM Corp. 1985, 1991

5.13.89 QueryExtension

name: **STRING8**
=>
present: **BOOL**
major-opcode: **CARD8**
first-event: **CARD8**
first-error: **CARD8**

QueryExtension determines if the extension named is present. If the extension is present, it is returned in the *major-opcode*. If the extension is not available, *major-opcode* returns a zero. Any minor-opcode and the request formats are specific to the extension. If the extension involves additional event types, the base event type code is returned. Otherwise, a zero is returned.

The format of the events is specific to the extension. If the extension involves additional error codes, the base error code is returned. Otherwise, a zero is returned. The format of additional data in the errors is specific to the extension.

The extension name, which is case-sensitive, should be in ISO Latin-1 encoding.

¦ Copyright IBM Corp. 1985, 1991

5.13.90 QueryFont

```
   font: FONTABLE
=>
   font-info: FONTINFO
   char-infos: LISTofCHARINFO

    where
       FONTINFO: [draw-direction: {LeftToRight, RightToLeft}
                     min-char-or-byte2, max-char-or-byte, max-byte1: CARD8
                     all-chars-exist: BOOL
                     default-char: CARD16
                     min-bounds: CHARINFO
                     max-bounds: CHARINFO
                     font-ascent: INT16
                     font-descent: INT16
                     properties: LISTofFONTPROP]
        FONTPROP: [name: ATOM
                      value: <32-bit-value>]
        CHARINFO: [left-side-bearing: INT16
                      right-side-bearing: INT16
                      character-width: INT16
                      ascent: INT16
                      descent: INT16
                      attributes: CARD16]

      Errors: Font
```

QueryFont returns logical information about a font. If a **gcontext** is given, the currently contained font is used.

The *draw-direction* is just a hint, indicating whether most *char-infos* have a positive (**LeftToRight**) or a negative (**RightToLeft**) *character-width* metric. The Core protocol defines no support for vertical text.

If *min-byte1* and *max-byte1* are both zero, then *min-char-or-byte2* specifies the linear character index corresponding to the first element of *char-infos* and *max-char-or-byte2* specifies the linear character index of the last element. If either *min-byte1* or *max-byte1* are non-zero, then both *min-char-or-byte2* and *max-char-or-byte2* will be less than 256, and the 2-byte character index values corresponding to *char-infos* element *N* (counting from zero) are

```
byte1 = N/D + min-byte1
byte2 = N-D + min-char-or-byte2
where
D = max-char-or-byte2 - min-char-or-byte2 + 1
/ = integer division
" = integer modulus
```

If *char-infos* has length zero, then *min-bounds* and *max-bounds* will be identical, and the effective *char-infos* is one filled with this *char-info*, of length

```
L = D * (max-byte1 - min-byte1 + 1)
```

All glyphs in the specified linear or matrix range have the same information as that given by *min-bounds* (and *max-bounds*). If *all-chars-exist* is **True**, then all characters in *char-infos* have non-zero bounding boxes.

¦ Copyright IBM Corp. 1985, 1991

The *default-char* specifies the character that will be used when an undefined or non-existent character is used. Note that *default-char* is a **CARD16** (not **CHAR2B**); for a font using 2-byte matrix format, the *default-char* has *byte1* in the most-significant byte, and *byte2* is in the least-significant byte. If the *default-char* itself specifies an undefined or non-existent character, then printing is not performed for an undefined or non-existent character.

The *min-bounds* and *max-bounds* contain the minimum and maximum values of each individual **CHARINFO** component over all *char-infos* (ignoring non-existent characters). The bounding box of the font (the smallest rectangle enclosing the shape obtained by superimposing all characters at the same origin [*x,y*]) has its upper-left coordinate at

```
[x + min-bounds.left-side-bearing, y - max-bounds.ascent]
```

with a width of

```
max-bounds.right-side-bearing - min-bounds.left-side-bearing
```

and a height of

```
max-bounds.ascent + max-bounds.descent
```

The *font-ascent* is the logical extent of the font above the baseline for determining line spacing. Specific characters can extend beyond this. The *font-descent* is the logical extent of the font at or below the baseline, for determining line spacing. Specific characters may extend beyond this. If the baseline is at Y-coordinate *y*, then the logical extent of the font is inclusive between the Y-coordinate values (*y* - *font-ascent*) and (*y* + *font-descent* - 1).

A font is not guaranteed to have any properties. The interpretation of the property value (such as, **INT32, CARD32**) must be derived from prior knowledge of the property. When possible, fonts should have at least the following properties. The following names are case-sensitive.

Property Name	Type	Description
MIN_SPACE	**CARD32**	The minimum inter-word spacing (in pixels).
NORM_SPACE	**CARD32**	The normal inter-word spacing (in pixels).
MAX_SPACE	**CARD32**	The maximum inter-word spacing (in pixels).
END_SPACE	**CARD32**	The additional spacing (in pixels) at the end of sentences.
SUPERSCRIPT_X SUPERSCRIPT_Y	**int**	Offset from the character origin where superscripts should begin (in pixels). If the origin is at [*x,y*], then superscripts should begin at [*x* + SUPERSCRIPT_X, *y* - SUPERSCRIPT_Y]
SUBSCRIPT_X SUBSCRIPT_Y	**INT32**	Offset from the character origin where subscripts should begin (in pixels). If the origin is at [*x,y*], then

¦ Copyright IBM Corp. 1985, 1991

		subscripts should begin at [*x* + SUPERSCRIPT_X, *y* + SUPERSCRIPT_Y].
UNDERLINE_POSITION	**INT32**	Y offset from the baseline to the top of an underline (in pixels). If the baseline is Y-coordinate *y*, then the top of the underline is at (*y* + UNDERLINE_POSITION).
UNDERLINE_THICKNESS	**CARD32**	Thickness of the underline (in pixels)
STRIKEOUT_ASCENT STRIKEOUT_DESCENT	**INT32**	Vertical extents for boxing or voiding characters (in pixels). If the baseline is at Y-coordinate *y*, then the top of the strikeout box is at (*y* - STRIKEOUT_ASCENT), and the height of the box is (STRIKEOUT_ASCENT + STRIKEOUT_DESCENT).
ITALIC_ANGLE	**INT32**	The angle of the dominant staffs of characters in the font, in degrees scaled by 64, relative to the three-o-clock position from the character origin, with positive indicating counterclockwise motion as in the **XDrawArc** functions.
X_HEIGHT	**INT32**	*1 ex* as in TeX, but expressed in units of pixels. Often the height of lowercase x.
QUAD_WIDTH	**INT32**	*1 em* as in TeX, but expressed in units of pixels. Often the width of the digits 0-9.
CAP_HEIGHT	**INT32**	Y offset from the baseline to the top of the capital letters, ignoring accents, in pixels. If the baseline is at Y-coordinate *y*, then the top of the capitals is at (*y* - CAP_HEIGHT).
WEIGHT	**CARD32**	The weight or boldness of the font, expressed as a value between 0 and 1000.
POINT_SIZE	**CARD32**	The point size of this font at the ideal resolution, expressed in 1/10ths of points. There are 72.27 points to the inch.
RESOLUTION	**CARD32**	The number of pixels per point, expressed in 1/100ths, at which this font was created.

The bounding box of a character is the smallest rectangle enclosing the shape of the character. If this bounding box has a character origin at [*x,y*] and is described in terms of **CHARINFO** components, then it is a rectangle with its upper-left corner at

¦ Copyright IBM Corp. 1985, 1991

[*x* + left-side-bearing, y - ascent]

and a width of

right-side-bearing - left-side-bearing

and a height of

ascent + descent

and the origin for the next character is defined to be

[*x* + character-width, *y*]

Note that the baseline is logically viewed as being just below non-descending characters (when *descent* is zero, only pixels with Y-coordinates less than *y* are drawn), and that the origin is logically viewed as being coincident with the left edge of a non-kerned character (when *left-side-bearing* is zero, no pixels with X-coordinate less than *x* are drawn).

Note that **CHARINFO** metric values can be negative.

A non-existent character is represented with all **CHARINFO** components zero.

The interpretation of the per-character attributes field is server-dependent.

¦ Copyright IBM Corp. 1985, 1991

5.13.91 QueryKeymap

=>

keys: **LISTofCARD8**

QueryKeymap returns a bit vector for the keyboard. Each one bit indicates that the corresponding key is currently pressed. The vector is represented as 32 bytes. Byte *N* (from 0) contains the bits for keys 8*N* to 8*N*+7, with the least significant bit in the byte representing key 8*N*.

¦ Copyright IBM Corp. 1985, 1991

5.13.92 QueryPointer

window: **WINDOW**
=>
root: **WINDOW**
child: **WINDOW** or **None**
same-screen: **BOOL**
root-x, root-y, win-x, win-y: **INT16**
mask: **SETofKEYBUTMASK**

Errors: **Window**

QueryPointer returns the *root* window the pointer is currently on and the pointer coordinates relative to the root origin. The current state of the modifier keys and the buttons are also returned.

If *same-screen* is **False**, then the pointer is not on the same screen as the argument *window*, *child* is **None** and *win-x*, and *win-y* are zero. If *same-screen* is **True**, then *win-x* and *win-y* are the pointer coordinates relative to the origin of the argument *window*, and *child* is the *child* containing the pointer, if any.

Note: The logical state of a device might lag behind the physical state if device event processing is frozen.

¦ Copyright IBM Corp. 1985, 1991

5.13.93 QueryTextExtents

```
   font: FONTABLE
   string: STRING16
=>
   draw-direction: {LeftToRight, RightToLeft}
   font-ascent: INT16
   font-descent: INT16
   overall-ascent: INT16
   overall-descent: INT16
   overall-width: 1INT32
   overall-left: INT32
   overall-right: INT32

      Errors: Font
```

QueryTextExtents returns the logical extents of the specified string of characters in the specified font. If a **gcontext** is given, the currently contained font is used.

The *draw-direction*, *font-ascent*, and *font-descent* fields are as described in **QueryFont**. See "QueryFont" in topic 5.13.90.

The *overall-ascent* field is the maximum of the ascent metrics of all characters in the string, and *overall-descent* is the maximum of the descent metrics.

The *overall-width* field is the sum of the character-width metrics of all characters in the string.

For each character in the *string*, let *W* be the sum of the character-width metrics of all characters preceding it in the *string*, let *L* be the left-side-bearing metric of the character plus *W* and let *R* be the right-side-bearing metric of the character plus *W*. The *overall-left* is the minimum *L* of all characters in the string, and *overall-right* is the maximum *R*.

For *fonts* defined with linear indexing rather than 2-byte matrix indexing, the server will interpret each **CHAR2B** as a 16-bit number that has been transmitted with the most significant byte first (*byte1* of the **CHAR2B** is taken as the most significant byte).

If the *font* has no defined default-char, then undefined characters in the *string* are taken to have all zero metrics.

¦ Copyright IBM Corp. 1985, 1991

5.13.94 QueryTree

window: **WINDOW**
=>
root: **WINDOW**
parent: **WINDOW** or **None**
children: **LISTofWINDOW**

Errors: **Window**

QueryTree returns the *root*, the *parent*, and *children* of the *window*. The children are listed in bottom-to-top stacking order.

¦ Copyright IBM Corp. 1985, 1991

5.13.95 RecolorCursor

curosr: **CURSOR**
fore-red, fore-green, fore-blue: **CARD16**
back-red, back-green, back-blue: **CARD16**

Errors: **Cursor**

RecolorCursor changes the color of a cursor. If the cursor is being displayed on a screen, the change is visible immediately.

¦ Copyright IBM Corp. 1985, 1991

5.13.96 ReparentWindow

window, parent: **WINDOW**
x, y: **INT16**

Errors: **Window, Match**

ReparentWindow removes a window from its current hierarchical position and inserts it as a child of the specified *parent*. The window is placed on top in the sibling stacking order. Then, a **ReparentNotify** event is generated.

If the window is mapped, an **UnmapWindow** is performed automatically so that the window is repositioned as a child of the *parent*.

The *x* and *y* coordinates, which are relative to the origin of the parent window, specify the new position of the upper-left outer corner of the window.

The *override-redirect* attribute of the window is passed in this event. If **True**, it indicates that a window manager should not tamper with this window. Finally, if the window was originally mapped, a **MapWindow** is performed automatically.

Normal exposure processing on formerly obscured windows is performed. The server might not generate exposure events for regions from the initial unmap that are immediately obscured by the final map.

A **Match** error is generated if:

The new parent is not on the same screen as the old parent

The new parent is the window or an inferior of the window

The window has a **ParentRelative** background and the new parent is not the same depth as the window.

¦ Copyright IBM Corp. 1985, 1991

5.13.97 RotateProperties

window: **WINDOW**
delta: **INT16**
properties: **LISTofATOM**

Errors: **Window, Atom, Match**

RotateProperties rotates the states of the properties.

If the property names in the *properites* list are numbered starting from zero and there are *N* property names in the list, then the value associated with property name *I* becomes the value associated with property name (*I* + *delta*) mod *N*, for all *I* from zero to *N* - 1. The effect is to rotate the states by *delta* places around the virtual ring of property names (right for positive *delta*, left for negative *delta*).

If *delta* mod *N* is non-zero, a **PropertyNotify** event is generated for each property in the order listed.

If an atom occurs more than once in the list or no property with that name is defined for the window, a **Match** error is generated. If an **Atom** or **Match** error is generated, no properties are changed.

¦ Copyright IBM Corp. 1985, 1991

5.13.98 SendEvent

destination: **WINDOW** or **PointerWindow** or **InputFocus**
propagate: **BOOL**
event-mask: **SETofEVENT**
event: <normal-event-format>

Errors: **Window, Value**

SendEvent sends an event to the *destination* window.

If **PointerWindow** is specified, *destination* is replaced with the window that the pointer is in. If **InputFocus** is specified and the focus window contains the pointer, *destination* is replaced with the window that the pointer is in; otherwise, *destination* is replaced with the focus window.

If the *event-mask* is the empty set, the *event* is sent to the client that created the destination window. If that client no longer exists, no *event* is sent.

If *propagate* is **False**, the *event* is sent to every client selecting any of the event types in *event-mask* on the *destination*.

If *propagate* is **True** and no clients selected any of the event types in *event-mask*, *destination* is replaced with the closest ancestor of *destination* for which some client has selected a type in *event-mask* and no intervening window has that type in its do-not-propagate-mask. If no such window exists or if the window is an ancestor of the focus window and **InputFocus** was originally specified as the *destination*, then the *event* is not sent to any clients. Otherwise, the *event* is reported to every client selecting on the final *destination* any of the types specified in *event-mask*.

The *event* code must be one of the Core events or one of the events defined by an extension so that the server can swap byte contents correctly, as necessary. Otherwise, the contents of *event* are unaltered and unchecked by the server except to force the most-significant bit of the *event* code.

Active grabs are ignored for this request.

¦ Copyright IBM Corp. 1985, 1991

5.13.99 SetAccessControl

mode: {**Enable, Disable**}

Errors: **Value, Access**

SetAccessControl enables or disables the use of the access control list at connection setups.

The client and the server must reside on the same host or the client must have permission by a server-dependent method to execute this request. Otherwise, an **Access** error occurs.

¦ Copyright IBM Corp. 1985, 1991

5.13.100 SetClipRectangles

gc: **GCONTEXT**
clip-x-origin, clip-y-origin: **INT16**
rectangles: **LISTofRECTANGLE**
ordering: {**UnSorted, YSorted, YXSorted, YXBanded**}

Errors: **GContext, Value, Alloc, Match**

SetClipRectangles changes clip-mask in *gc* to the specified list of *rectangles* and sets the clip origin. Output is clipped to remain within the *rectangles*. The clip origin is relative to the origin of the destination drawable specified in a graphics request. The rectangle coordinates are relative to the origin of the clip. The rectangles should be non-intersecting, otherwise graphics results are undefined. If the list of *rectangles* is empty, it disables output effectively. This is opposite to passing **None** as the clip-mask in **CreateGC** and **ChangeGC**.

If the client knows the order of the *rectangles*, it should be specified with the *ordering* argument. By setting the ordering, the server may operate faster. If the ordering is specified incorrectly, the server can generate a **Match** error. If no error is generated, the graphics results are undefined.

The *ordering* argument can be:

UnSorted which indicates that the rectangles are in an arbitrary order.

YSorted which indicates that the rectangles are non-decreasing in their Y-axis origin.

YXSorted, which constrains the **YSorted** order, inidcates that all rectangles with an equal Y-axis origin are non-decreasing in their origin.

YXBanded, which constrains **YXSorted**, requires that all rectangles with the Y-axis scanline should have identical Y-axis origins and Y-axis extents.

¦ Copyright IBM Corp. 1985, 1991

5.13.101 SetCloseDownMode

mode: {**Destroy, RetainPermanent, RetainTemporary**}

Errors: **Value**

SetCloseDownMode defines what happens to the client resources at a connection close. A connection starts in **Destroy** mode. See "Closing Connections to the Server" in topic 5.10 for information about the **close-down** mode.

¦ Copyright IBM Corp. 1985, 1991

5.13.102 SetDashes

gc: **GCONTEXT**
dash-offset: **CARD16**
dashes: **LISTofCARD8**

Errors: **GContext, Value, Alloc**

SetDashes sets *dash-offset* and dashes in *gc* for dashed line styles. The *dashes* cannot be empty. Specifying an odd-length list is equivalent to specifying the same list concatenated with itself to produce an even-length list.

The *even dashes* are the initial and alternating elements.

The *odd dashes* are all the other dashes.

All of the elements must be a non-zero.

The *dash-offset* defines the phase of the pattern specifying how many pixels into *dashes* the pattern should actually begin in any single graphics request. Dashing is continuous through path elements combined with a join-style, but is reset to the *dash-offset* each time a cap-style is used at a line endpoint.

The unit of measure for *dashes* is the same as in the ordinary coordinate system. Ideally, a dash length is measured along the slope of the line, but implementations are only required to match this ideal for horizontal and vertical lines. Failing the ideal semantics, it is suggested that the length be measured along the major axis of the line. The major axis is defined as the X-axis for lines drawn at an angle of between - 45 and + 45 degrees or between 315 and 225 degrees from the X-axis. For all other lines, the major axis is the Y-axis.

¦ Copyright IBM Corp. 1985, 1991

5.13.103 SetFontPath

path: **LISTofSTRING8**

Errors: **Value**

SetFontPath defines the path used to search for fonts. It sets a list of directories for the server to use to search for font files. These directories are operating-system specific.

Only one search path per server exists and not one per client. The interpretation of the strings is operating-system dependent, but the strings are intended to specify directories to be searched in the order listed. Setting the *path* to the empty list restores the default path defined for the server.

As a side-effect of executing this request, the server is guaranteed to flush all cached information about fonts for which no explicit resource IDs allocated currently exist.

The meaning of an error from this request is system-specific.

¦ Copyright IBM Corp. 1985, 1991

5.13.104 SetInputFocus

focus: **WINDOW** or **PointerRoot** or **None**
revert-to: {**Parent, PointerRoot, None**}
time: **TIMESTAMP** or **CurrentTime**

Errors: **Window, Value, Match**

SetInputFocus changes the input focus and the last-focus-change time. The request has no effect if the specified *time* is earlier than the current last-focus-change *time* or is later than the current server *time*. Otherwise, the last-focus-change *time* is set to the specified *time*, with **CurrentTime** replaced by the current server *time*.

If **None** is specified as the *focus*, all keyboard events are discarded until a new focus window is set. In this case, the *revert-to* argument is ignored.

If a **WINDOW** is specified as the *focus*, it becomes the focus window of the keyboard. If a keyboard event is reported to this window or one of its inferiors normally, this window still receives it. Otherwise, the event is reported with respect to the *focus* window.

If **PointerRoot** is specified as the *focus*, the focus window is taken dynamically to be the root window of the screen that contains the pointer during each keyboard event. In this case, the *revert-to* argument is ignored.

This request generates **FocusIn** and **FocusOut** events.

The specified *focus* window must be viewable at the time of the request. Otherwise, a **Match** error occurs. If the *focus* window becomes unviewable later, the new focus window depends on the *revert-to* argument.

If *revert-to* is **Parent**, the *focus* reverts to the parent, or the closest viewable ancestor and *revert-to* becomes **None**.

If *revert-to* is **PointerRoot** or **None**, the *focus* reverts to that value.

When the *focus* reverts, **FocusIn** and **FocusOut** events are generated, but the last-focus-change time is not affected.

¦ Copyright IBM Corp. 1985, 1991

5.13.105 SetModifierMapping

keycodes-per-modifier: **CARD8**
keycodes: **LISTofKEYCODE**
=>
status: {**Success, Busy, Failed**}

Errors: **Value, Alloc**

SetModifierMapping specifies the *keycodes* (if any) to be used as modifiers.

The number of *keycodes* in the list must be **8*keycodes-per-modifier**, otherwise a **Length** error occurs.

The *keycodes* are divided into eight sets, with each set containing *keycodes-per-modifier* elements. The sets are assigned to the following modifiers in order: **Shift, Lock, Control, Mod1, Mod2, Mod3, Mod4, and Mod5**. Only non-zero keycode values are used within each set; zero values are ignored. All non-zero *keycodes* must be in the range specified by *min-keycode* and *max-keycode* at connection setup, otherwise a **Value** error occurs.

The order of *keycodes* within a set is not significant. If non-zero values are not specified, the use of the corresponding modifier is disabled, and the modifier bit will be a value of zero. Otherwise, when at least one of the keys in the corresponding set is in the *down* position, the modifier bit will be a value of one.

A server can impose restrictions on how modifiers can be changed, for example, if certain keys do not generate *up* transitions in hardware or if multiple keys per modifier are not supported.

If *status* is:

Failed, a restriction is violated and no modifier is changed.

Busy, no modifiers are changed because the new non-zero *keycodes* specified for a modifier are different from the defined *keycodes* and any (current or new) keys for that modifier are in the *down* state.

Success, modifiers are changed and a **MappingNotify** event is generated.

¦ Copyright IBM Corp. 1985, 1991

5.13.106 SetPointerMapping

map: **LISTofCARD8**
=>
status: {**Success, Busy**}

Errors: **Value**

SetPointerMapping sets the mapping of the pointer. The elements of the mapping list are indexed starting with one. The length of the list must be the same the list returned by **GetPointerMapping**. The index is a Core button number and the element of the list defines the effective number.

A zero element disables a button. Elements are not restricted in value by the number of physical buttons, but no two elements can have the same non-zero value.

If *status* is:

Busy, the mapping is not changed because the buttons to be altered are in the *down* state.

Success, the mapping is changed and a **MappingNotify** event is generated.

¦ Copyright IBM Corp. 1985, 1991

5.13.107 SetScreenSaver

timeout, interval: **INT16**
prefer-blanking: {**Yes, No, Default**}
allow-exposures: {**Yes, No, Default**}

Errors: **Value**

SetScreenSaver specifies *timeout* and *interval* in seconds. If *timeout* is:

A zero, the screen-saver is disabled

A non-zero, the screen-saver is enabled. Once enabled, th screen-saver is activated if the keyboard or pointer do not receive input within the *timeout* period.

To restore the default, set the the *timeout* value to -1. Other negative values generate an error.

For each screen, if blanking is preferred and the hardware supports video blanking, the screen goes blank. Otherwise, if exposures are allowed or the screen can be regenerated without sending exposure events to clients, the screen is changed to avoid phosphor burn. Otherwise, the state of the screens does not change and screen-saver is not activated. Screen-saver is deactivated and all screen states are restored with the next keyboard or pointer input or at the next **ForceScreenSaver** that has mode set to **Reset**.

If the screen-saver can be changed, the *interval* serves as a hint about how long the change period should be. A value of zero hints that no periodic change should be made.

Examples of ways to change the screen include scrambling the colormap periodically, moving an icon image about the screen periodically, or tiling the screen with the root window background tile that is re-started randomly and periodically.

¦ Copyright IBM Corp. 1985, 1991

5.13.108 SetSelectionOwner

selection: **ATOM**
owner: **WINDOW** or **None**
time: **TIMESTAMP** or **CurrentTime**

Errors: **Atom, Window**

SetSelectionOwner changes the owner, owner window, and last-change time of the specified *selection*. The *selection* atom is not interpreted by the server. The owner window is returned by the **GetSelectionOwner** It is reported in **SelectionRequest** and **SelectionClear** events. Selections are global to the server.

The *owner* becomes the client executing the request, unless **None** is specified. If **None** is specified, the window does not have an owner.

If the new *owner* is not the same as the current *owner* and the current *owner* is not **None**, then the current *owner* is sent a **SelectionClear** event.

If *owner* is a client and that client is terminated later, this connection is closed. If *owner* is a window and that window is destroyed later, then the *owner* reverts automatically to **None**, and the last-change *time* is not affected.

This request has no effect if the specified *time* is earlier than the current last-change time of the specified *selection* or is later than the current server time. Otherwise, the last-change time is set to the specified *time* and **CurrentTime** is replaced by the current server time.

¦ Copyright IBM Corp. 1985, 1991

5.13.109 StoreColors

cmap: **COLORMAP**
items: **LISTofCOLORITEM**

where

COLORITEM: [*pixel*: **CARD32**
do-red, do-green, do-blue: **BOOL**
red, green, blue: **CARD16**]

Errors: **Colormap, Access, Value**

StoreColors changes the colormap entries of the specified pixels. If the colormap for its screen is an installed map, the changes are visible immediately.

The *do-red, do-green,* and *do-blue* indicate which components should be changed.

All specified pixels that are allocated writable in *cmap* (by any client) are changed, even if one or more pixels produce an error.

If a specified *pixel* is not a valid index in the specified colormap, a **Value** error is generated.

If a specified *pixel* is unallocated or allocated read-only, an **Access** error is generated.

¦ Copyright IBM Corp. 1985, 1991

5.13.110 StoreNamedColor

cmap: **COLORMAP**
pixel: **CARD32**
name: **STRING8**
do-red, do-green, do-blue **BOOL**

Errors: **Colormap, Name, Access, Value**

StoreNamedColor searches for the color name for the screen associated with the specified colormap. Then, it stores the color name in the specified colormap. (The color name is not case-sensitive or an ASCII string.)

If a specified *pixel* is not a valid index in the specified colormap, a **Value** error is generated.

If a specified *pixel* is unallocated or allocated read-only, an **Access** error is generated.

¦ Copyright IBM Corp. 1985, 1991

5.13.111 TranslateCoordinates

```
   src-window, dst-window: WINDOW
   src-x, src-y: INT16
=>
   same-screen: BOOL
   child: WINDOW or None
   dst-x, dst-y: INT16

      Errors: Window
```

TranslateCoordinates returns *dst-x* and *dst-y* coordinates to the origin of the *dst-window* as these coordinates relate to the *src-x* and *src-y* coordinates in relationship to the origin of the *src-window*.

If *same-screen* is **False**, the *src-window* and *dst-window* are on different screens, and *dst-x* and *dst-y* are zero. If the coordinates are in a mapped child of *dst-window*, then that child is returned.

¦ Copyright IBM Corp. 1985, 1991

5.13.112 UngrabButton

modifiers: **SETofKEYMASK** or **AnyModifier**
button: **BUTTON** or **AnyButton**
grab-window: **WINDOW**

Errors: **Window**

UngrabButton releases the passive button-key combination on the specified window if it was grabbed by this client. It has no effect on an active grab.

A *modifiers* of **AnyModifier** is equivalent to issuing the request for all possible modifier combinations, including the combination of no *modifiers*. A *button* of **AnyButton** is equivalent to issuing the request for all possible buttons.

¦ Copyright IBM Corp. 1985, 1991

5.13.113 UngrabKey

key: **KEYCODE** or **AnyKey**
modifiers: **SETofKEYMASK** or **AnyModifier**
grab-window: **WINDOW**

Errors: **Window**

UngrabKey releases the key combination on the specified window if the key combination was grabbed by this client. It has no effect on an active grab.

A *modifiers* of **AnyModifier** is equivalent to issuing the request for all possible modifier combinations (including the combination of no *modifiers*).

A key of **AnyKey** is equivalent to issuing the request for all possible keycodes.

¦ Copyright IBM Corp. 1985, 1991

5.13.114 UngrabKeyboard

time: **TIMESTAMP** or **CurrentTime**

UngrabKeyboard releases the keyboard if the client has grabbed the keyboard actively with **GrabKeyboard** or **GrabKey**. It releases any queued events. It generates **FocusIn** and **FocusOut** events.

UngrabKeyboard is performed automatically if the event window for an active keyboard grab becomes unviewable.

UngrabKeyboard has no effect if the specified *time* is earlier than the last-keyboard-grab time or is later than the current server *time*.

¦ Copyright IBM Corp. 1985, 1991

5.13.115 UngrabPointer

time: **TIMESTAMP** or **CurrentTime**

UngrabPointer releases the pointer if this client has grabbed the pointer actively with **GrabPointer**, **GrabButton**, or a normal button press. It releases any queued events. It generates **EnterNotify** and **LeaveNotify** events.

UngrabPointer is performed automatically if the event or confine-to window for an active pointer grab becomes unviewable.

UngrabPointer has no effect if the specified *time* is earlier than the last-pointer-grab time or is later than the current time.

¦ Copyright IBM Corp. 1985, 1991

5.13.116 UngrabServer

UngrabServer restarts processing of requests and close-downs on other connections.

¦ Copyright IBM Corp. 1985, 1991

5.13.117 UninstallColormap

cmap: **COLORMAP**

Errors: **Colormap**

UninstallColormap removes the specified colormap if the colormap is on the required list for the screen. (See "InstallColormap" in topic 5.13.63.) The required list must remain installed. Incidentially, it can also install or uninstall additional colormaps depending on the server.

When the *cmap* is uninstalled, a **ColormapNotify** event is generated on each window with this *cmap* as an attribute. In addition, a **ColormapNotify** event is generated if other colormaps are installed or uninstalled as a result of this request.

¦ Copyright IBM Corp. 1985, 1991

5.13.118 UnmapSubwindows

window: **WINDOW**

Errors: **Window**

UnmapSubwindows performs an **UnmapWindow** request on all mapped children of the window in bottom-to-top stacking order.

¦ Copyright IBM Corp. 1985, 1991

5.13.119 UnmapWindow

window: **WINDOW**

Errors: **Window**

UnmapWindow unmaps a mapped window and generates an **UnmapNotify** event. Normal exposure processing on formerly obscured windows is performed.

If the window is already unmapped, this request has no effect.

¦ Copyright IBM Corp. 1985, 1991

5.13.120 WarpPointer

src-window: **WINDOW** or **None**
dst-window: **WINDOW** or **None**
src-x, src-y: **INT16**
src-width, src-height: **CARD16**
dst-x, dst-y: **INT16**

Errors: **Window**

WarpPointer moves the current position of the pointer. It generates events as if the user had moved the pointer instantaneously.

If *dst-window* is **None**, it moves the pointer to the position indicated by *dst-x, dst-y* coordinates. These coordinates are relative to the current position of the pointer.

If *dst-window* is a window, it moves the pointer to the position indicated by *dst-x, dst-y* coordinates. These coordinates are relative to the origin of the *dst-window*.

If *src-window* is not **None**, the move takes place only if the pointer is currently contained in a visible portion of the specified rectangle of the *src-window*.

The *src-x* and *src-y* coordinates are relative to the origin of the *src-window*.

If *src-height* is zero, it is replaced with the current height of *src-window* minus *src-y*.

If *src-width* is zero, it is replaced with the current width of *src-window* minus *src-x*.

This request cannot be used to move the pointer outside the confine-to window of an active pointer grab because it moves the pointer only as far as the closest edge of the confine-to window.

¦ Copyright IBM Corp. 1985, 1991

5.14 Events

This section contains detailed descriptions of events. Events are arranged in alphabetical order.

Subtopics

¦ Copyright IBM Corp. 1985, 1991

5.14.1 CirculateNotify

event, window: **WINDOW**
place: {**Top, Bottom**}

CirculateNotify event is reported to clients selecting **StructureNotify** on the window and to clients selecting **SubstructureNotify** on the parent. It is generated when the window is restacked as a result of a **CirculateWindow** request.

The *event* is the window on which the event is generated.

The *window* is the window to be restacked.

The *place* is the position of the window within the stack.

If *place* is **Top**, the window is placed on top of all siblings. Otherwise, it is placed below all siblings.

¦ Copyright IBM Corp. 1985, 1991

5.14.2 CirculateRequest

parent, window: **WINDOW**
place: {**Top, Bottom**}

CirculateRequest event is reported to the client selecting **SubstructureRedirect** on the parent. It is generated when a **CirculateWindow** request is issued on the *parent*, and a *window* needs to be restacked.

The *window* is the window to be restacked.

The *place* is the position of the window in the stack.

¦ Copyright IBM Corp. 1985, 1991

5.14.3 ClientMessage

window: **WINDOW**
type: **ATOM**
format: {**8, 16, 32**}
data: **LISTofINT8** or **LISTofINT16** or **LISTofINT32**

ClientMessage event is generated only by clients using **SendEvent**.

The *type* specifies how the data is to be interpreted by the receiving client.

The *format* specifies whether the data should be in 8-bit, 16-bit, or 32-bit quantities.

The *data* consists of twenty 8-bit values or ten 16-bit values or five 32-bit values, although particular message *types* may not use all these values.

¦ Copyright IBM Corp. 1985, 1991

5.14.4 ColormapNotify

window: **WINDOW**
colormap: **COLORMAP** or **None**
new: **BOOL**
state: {**Installed, Uninstalled**}

ColormapNotify event is reported to clients selecting **ColormapChange** on the window. This event is generated under the following conditions:

When the *colormap* is changed and the attribute *new* is **True**.

When the colormap is installed or uninstalled and the attribute *new* is **False**.

In either case, *state* indicates if the colormap is installed currently.

¦ Copyright IBM Corp. 1985, 1991

5.14.5 ConfigureNotify

event, window: **WINDOW**
x, y: **INT16**
width, height, border-width: **CARD16**
above-sibling: **WINDOW** or **None**
override-redirect: **BOOL**

ConfigureNotify event is reported to clients selecting **StructureNotify** on the window, and to clients selecting **SubstructureNotify** on the parent. It is generated when a **ConfigureWindow** request changes the state of the window.

The *event* is the window on which the event is generated.

The *window* is the window to be changed.

The *x* and *y* coordinates, which are relative to the origin of the new parent window, specify the position of the upper-left outer corner of the window.

The *width* and *height* specify the inside of the window, excluding the border.

The *above-sibling* is the position of the window among the sibling windows.

If *above-sibling* is **None**, the window is at the bottom of the stack with respect to siblings. Otherwise, the window is immediately on top of the specified sibling.

The *override-redirect* is taken from the window attribute.

¦ Copyright IBM Corp. 1985, 1991

5.14.6 ConfigureRequest

parent, window: **WINDOW**
x, y: **INT16**
width, height, border-width: **CARD16**
sibling: **WINDOW** or **None**
stack-mode: {**Above, Below, TopIf, BottomIf, Opposite**}
value-mask: **BITMASK**

ConfigureRequest event is reported to the client selecting **SubstructureRedirect** on the parent. It is generated when a **ConfigureWindow** request is issued (by another client) on the window.

The *value-mask* and the corresponding values are reported as given in the request. The *sibling* is **None**, if not specified in the request. The *stack-mode* is **Above**, if not specified in the request. The other values are taken from the current geometry of the window.

¦ Copyright IBM Corp. 1985, 1991

5.14.7 CreateNotify

parent, window: **WINDOW**
x, y: **INT16**
width, height, border-width: **CARD16**
override-redirect: **BOOL**

CreateNotify event is reported to clients selecting **SubstructureNotify** on the parent. It is generated when the window is created. See "CreateWindow" in topic 5.13.30 for a discussion of the arguments.

¦ Copyright IBM Corp. 1985, 1991

5.14.8 DestroyNotify

event, window: **WINDOW**

DestroyNotify event is reported to clients selecting **StructureNotify** on the window and to clients selecting **SubstructureNotify** on the parent. It is generated when the window is destroyed.

The *event* is the window on which the event is generated.

The *window* is the window to be destroyed.

A **DestroyNotify** event on any window is generated on all inferiors of the window before it is generated on the window. Otherwise, the ordering among siblings and across sub-hierarchies is not constrained.

¦ Copyright IBM Corp. 1985, 1991

5.14.9 EnterNotify, LeaveNotify

root, event: **WINDOW**
child:**WINDOW** or **None**
same-screen: **BOOL**
root-x, root-y, event-x, event-y: **INT16**
mode: {**Normal, Grab, Ungrab**}
detail: {**Ancestor, Virtual, Inferior, Nonlinear, NonlinearVirtual**}
focus: **BOOL**
state: **SETofKEYBUTMASK**
time:**TIMESTAMP**

EnterNotify and **LeaveNotify** events are generated if a pointer motion or window hierarchy change causes the pointer to be in a different window.

Only clients selecting **EnterWindow** on a window receive **EnterNotify** events. Only clients selecting **LeaveWindow** receive **LeaveNotify** events. The position of the pointer reported in the event is always the final position of the pointer, not the initial position of the pointer.

The *root* is the root window for this position. The *root-x* and *root-y* are the coordinates of the pointer relative to the origin of the *root* at the time of the event.

The *event* is the event window.

If the event window is on the same screen as *root*, then *event-x* and *event-y* coordinates of the pointer are relative to the origin of the event window. Otherwise, *event-x* and *event-y* are zero.

In a **LeaveNotify** event, if a child of the event window contains the initial position of the pointer, then the *child* component is set to that child. Otherwise, the *child* component is **None**.

In an **EnterNotify** event, if a child of the event window contains the final pointer position, then the *child* component is set to that child. Otherwise, the *child* component is **None**.

The *mode* is set to **Normal** for normal pointer motion events. The *mode* for pseudo-motion events is **Grab**. When a grab deactivates, *mode* is **Ungrab** for pseudo-motion events.

The *focus* is **True** if the event window is the focus window or an inferior of the focus window. Otherwise, *focus* is **False**.

EnterNotify and **LeaveNotify** events caused by a hierarchy change are generated after the hierarchy event (**UnmapNotify, MapNotify, ConfigureNotify, GravityNotify, CirculateNotify**) caused by that change. The ordering of **EnterNotify** and **LeaveNotify** events with respect to **FocusOut**, **VisibilityNotify**, and **Expose** events is not constrained.

Normal events are generated according to the following scenarios:

When the pointer moves from window *A* to window *B* and *A* is an inferior of *B*:

A **LeaveNotify** event (with *detail* **Ancestor**) is generated on *A*.

A **LeaveNotify** event (with *detail* **Virtual**) is generated on each window

¦ Copyright IBM Corp. 1985, 1991

between *A* and *B* exclusive, in that order.

An **EnterNotify** event (with *detail* **Inferior**) is generated on *B*.

When the pointer moves from window *A* to window *B* and *B* is an inferior of *A*:

A **LeaveNotify** event (with *detail* **Inferior**) is generated on *A*.

An **EnterNotify** event (with *detail* **Virtual**) is generated on each window between *A* and *B* exclusive, in that order.

An **EnterNotify** event (with *detail* **Ancestor**) is generated on *B*.

When the pointer moves from window *A* to window *B* with window *C* being their least common ancestor:

A **LeaveNotify** event (with *detail* **Nonlinear**) is generated on *A*.

A **LeaveNotify** event (with *detail* **NonlinearVirtual**) is generated on each window between *A* and *C* exclusive, in that order.

An **EnterNotify** event (with *detail* **NonlinearVirtual**) is generated on each window between *C* and *B* exclusive, in that order.

An **EnterNotify** event (with *detail* **Nonlinear**) is generated on *B*.

When the pointer moves from window A to window *B* on different screens:

A **LeaveNotify** event (with *detail* **Nonlinear**) is generated on *A*.

If *A* is not a root window, a **LeaveNotify** event (with *detail* **NonlinearVirtual**) is generated on each window above *A* up to and including its root, in order.

If *B* is not a root window, an **EnterNotify** event (with *detail* **NonlinearVirtual**) is generated on each window from the root of *B* down to but not including *B*, in that order.

An **EnterNotify** event (with *detail* **Nonlinear**) is generated on *B*.

When a pointer grab activates (after any initial warp into a confine-to window and before a **ButtonPress** event that activates the grab) with *G* (grab-window for the grab) and *P* (window where the pointer is located), **EnterNotify** and **LeaveNotify** events (with *mode* **Grab**) are generated as if the pointer suddenly warped from its current position in *P* to some position in *G*. However, the pointer does not warp and the pointer position is used as both the initial pointer position and the final pointer position for the events.

When a pointer grab deactivates (after generating a **ButtonRelease** event that deactivates the grab) with *G* (grab-window for the grab) and *P* (window where the pointer is located), **EnterNotify** and **LeaveNotify** events (with *mode* **Ungrab**) are generated as if the pointer suddenly warped from some position in *G* to its current position in *P*. However, the pointer does not warp and the current pointer position is used as both the initial pointer position and the final pointer position for the events.

¦ Copyright IBM Corp. 1985, 1991

5.14.10 Expose

window: **WINDOW**
x, y, width, height: **CARD16**
count: **CARD16**

Expose event is reported to clients selecting **Exposure** on the window. It can be generated when a region of the window becomes viewable, but might be generated only when a region becomes visible. All regions exposed by a given action are guaranteed to be reported contiguously.

Expose events are not generated on **InputOnly** windows.

The *x* and *y* coordinates, which are relative to the origin of the drawable, specify the upper-left corner of a rectangle.

The *width* and *height* specify the extent of the rectangle.

If *count* is zero, no more **Expose** events for this window will follow. If *count* is non-zero, then at least that many, and possibly more, **Expose** events for this window will follow.

All **Expose** events caused by a hierarchy change are generated after the hierarchy event (**UnmapNotify, MapNotify, ConfigureNotify, GravityNotify, CirculateNotify**) caused by that change.

Expose events on a window are generated after any **VisibilityNotify** event on that window, but it is not required that all **Expose** events on all windows be generated after all **Visibility** events on all windows. The ordering of **Expose** events with respect to **FocusOut**, **EnterNotify**, and **LeaveNotify** events is not constrained.

¦ Copyright IBM Corp. 1985, 1991

5.14.11 FocusIn, FocusOut

event: **WINDOW**
mode: {**Normal, WhileGrabbed, Grab, Ungrab**}
detail: {**Ancestor, Virtual, Inferior, Nonlinear, NonlinearVirtual, Pointer**
PointerRoot, None}

FocusIn and **FocusOut** events are reported to clients selecting **FocusChange** on the window. These events are generated when the input focus changes

The *mode* can be one of the following:

Normal if the events were generated by **SetInputFocus** when the keyboard is not grabbed.

WhileGrabbed if the events were generated by **SetInputFocus** when the keyboard is grabbed.

Ungrab if the events were generated when a keyboard grab activated.

Ungrab if the events were generated when a keyboard grab deactivated.

FocusOut events caused by a window unmap are generated after an **UnmapNotify** event. Otherwise, the order of **FocusOut** events, with respect to **EnterNotify**, **LeaveNotify**, **VisibilityNotify**, and **Expose** events, is not constrained.

Normal and **WhileGrabbed** events are generated according to the following scenarios:

When the focus moves from window *A* to window *B* and *A* is an inferior of *B* with the pointer in window *P*:

A **FocusOut** event (with *detail* **Ancestor**) is generated on *A*.

A **FocusOut** event (with *detail* **Virtual**) is generated on each window between *A* and *B* exclusive, *A* and *B* excluded (in that order).

A **FocusIn** event (with *detail* **Inferior**) is generated on *B*.

If *P* is an inferior of *B*, but *P* is not *A* or an inferior of *A* or an ancestor of *A*, a **FocusIn** event (with *detail* **Pointer**) is generated on each window below *B* down to and including *P* (in order).

When the focus moves from window *A* to window *B* and *B* is an inferior of *A* with the pointer in window *P*:

If *P* is an inferior of *A*, but *P* is not *A* or an inferior of *B* or an ancestor of *B*, a **FocusOut** event (with *detail* **Pointer**) is generated on each window from *P* up to but not including *A* (in order).

A **FocusOut** event (with *detail* **Inferior**) is generated on *A*.

A **FocusIn** event (with *detail* **Virtual**) is generated on each window between *A* and *B* exclusive.

A **FocusIn** event (with *detail* **Ancestor**) is generated on *B*.

When the focus moves from window *A* to window *B* with window *C* being their least common ancestor and with the pointer in window *P*:

¦ Copyright IBM Corp. 1985, 1991

If *P* is an inferior of *A*, a **FocusOut** event (with *detail* **Pointer**) is generated on each window from *P* up to but not including *A* (in order).

A **FocusOut** event (with *detail* **Nonlinear**) is generated on *A*.

A **FocusOut** event (with *detail* **NonlinearVirtual**) is generated on each window between *A* and *C* exclusive (in that order).

A **FocusIn** event (with *detail* **NonlinearVirtual**) is generated on each window between *C* and *B* exclusive (in that order).

A **FocusIn** event (with *detail* **Nonlinear**) is generated on *B*.

If *P* is an inferior of *B*, a **FocusIn** event (with *detail* **Pointer**) is generated on each window below *B* down to and including *P* (in order).

When the focus moves from window *A* to window *B* on different screens, with the pointer in window *P*:

If *P* is an inferior of *A*, a **FocusOut** event (with *detail* **Pointer**) is generated on each window from *P* up to but not including *A* (in order).

A **FocusOut** event (with *detail* **Nonlinear**) is generated on *A*.

If *A* is not a root window, a **FocusOut** event (with *detail* **NonlinearVirtual**) is generated on each window above *A* up to and including its root (in order).

If *B* is not a root window, a **FocusIn** event (with *detail* **NonlinearVirtual**) is generated on each window from the root of *B* down to but not including *B* (in order).

A **FocusIn** event (with *detail* **Nonlinear**) is generated on *B*.

If *P* is an inferior of *B*, a **FocusIn** event (with *detail* **Pointer**) is generated on each window below *B* down to and including *P* (in order).

When the focus moves from window *A* to **PointerRoot** or **None** with the pointer in window *P*:

If *P* is an inferior of *A*, a **FocusOut** event (with *detail* **Pointer**) is generated on each window from *P* up to but not including *A* (in order).

A **FocusOut** event (with *detail* **Nonlinear**) is generated on *A*.

If *A* is not a root window, a **FocusOut** event (with *detail* **NonlinearVirtual**) is generated on each window above *A* up to and including its root (in order).

A **FocusIn** event (with *detail* **PointerRoot** or **None**) is generated on all root windows.

If the new focus is **PointerRoot**, a **FocusIn** event (with *detail* **Pointer**) is generated on each window from the root of *P* down to and including *P* (in order).

When the focus moves from **PointerRoot** or **None** to window *A* with the pointer in window *P*:

¦ Copyright IBM Corp. 1985, 1991

If the old focus is **PointerRoot**, a **FocusOut** event (with *detail* **Pointer**) is generated on each window from *P* up to and including the root of *P* (in order).

A **FocusOut** event (with *detail* **PointerRoot** or **None**) is generated on all root windows.

If *A* is not a root window, a **FocusIn** event (with *detail* **NonlinearVirtual**) is generated on each window from the root of *A* root down to but not including *A* (in order).

A **FocusIn** event (with *detail* **Nonlinear**) is generated on *A*.

If *P* is an inferior of *A*, a **FocusIn** event (with *detail* **Pointer**) is generated on each window below *A* down to and including *P* (in order).

When the focus moves from **PointerRoot** to **None** (or vice versa) with the pointer in window *P*:

If the old focus is **PointerRoot**, a **FocusOut** event (with *detail* **Pointer**) is generated on each window from *P* up to and including the root of *P* (in order).

A **FocusOut** event (with *detail* **PointerRoot** or **None**) is generated on all root windows.

A **FocusIn** event (with *detail* **None** or **PointerRoot**) is generated on all root windows.

If the new focus is **PointerRoot**, a **FocusIn** event (with *detail* **Pointer**) is generated on each window from the root of *P* down to and including *P* (in order).

When a keyboard grab activates (before generating a **KeyPress** event that activates the grab) with *G* (grab-window for the grab) and *F* (current focus):

FocusIn and **FocusOut** events with *mode* **Grab** are generated as if the focus was changed from *F* to *G*.

When a keyboard grab deactivates (after generating a **KeyRelease** event that deactivates the grab) with *G* (grab-window for the grab) and *F* (the current focus):

FocusIn and **FocusOut** events with *mode* **Ungrab** are generated as if the focus was changed from *G* to *F*.

¦ Copyright IBM Corp. 1985, 1991

5.14.12 GraphicsExposure

drawable: **DRAWABLE**
x, y, width, height: **CARD16**
count: **CARD16**
major-opcode: **CARD8**
minor-opcode: **CARD16**

GraphicsExposure event is reported to clients selecting graphics exposures in a graphics context. It is generated when a destination region can not be computed due to an obscured or out-of-bounds source region. All regions exposed by a given graphics request are guaranteed to be reported contiguously.

The *x* and *y* coordinates, which are relative to the origin of the *drawable* origin, specify the upper-left corner of a rectangle.

The *width* and *height* specify the extent of the rectangle.

The *count* specifies the number of graphics exposures events that should follow.

If *count* is zero, no **GraphicsExposure** events will follow. If *count* is a non-zero, at least that number, but possibly more, **GraphicsExposure** events will follow.

The *major-opcode* and *minor-opcode* identify the graphics request. For the Core protocol, *major-opcode* is **CopyArea** or **CopyPlane** and *minor-opcode* is zero.

¦ Copyright IBM Corp. 1985, 1991

5.14.13 GravityNotify

event, window: **WINDOW**
x, y: **INT16**

GravityNotify event is reported to clients selecting **SubstructureNotify** on the *parent* and to clients selecting **StructureNotify** on the window. It is generated when a window is moved as a result of resizing the parent window.

The *event* is the window on which the event is generated.

The *window* is the window to be moved.

The *x* and *y* coordinates, which are relative to the origin of the new parent, specify the position of the upper-left outer corner of the window.

¦ Copyright IBM Corp. 1985, 1991

5.14.14 KeymapNotify

keys: **LISTofCARD8**

KeymapNotify event is reported to clients selecting **KeymapState** on a window. It is generated immediately after every **EnterNotify** event and **FocusIn** event. The value of *keys* is a bit vector as described in **QueryKeymap**. (See "QueryKeymap" in topic 5.13.91.)

¦ Copyright IBM Corp. 1985, 1991

5.14.15 KeyPress, KeyRelease, ButtonPress, ButtonRelease, MotionNotify

root, event: **WINDOW**
child: **WINDOW** or **None**
same-screen: **BOOL**
root-x, root-y, event-x, event-y: **INT16**
detail: <see below>
state: **SETofKEYBUTMASK**
time: **TIMESTAMP**

KeyRelease, ButtonRelease, ButtonPress, and **MotionNotify** events are generated when a key or button changes state or the pointer moves. **KeyPress** and **KeyRelease** are generated for all keys, even those mapped to modifier bits.

The source of the event is the window in which the pointer is located. The window to which the event normally is reported is found by searching the hierarchy (beginning with the source window). It is the first window on which any client has selected interest in the event, if no other window prohibits event generation by including the event type in its do-not-propagate-mask. The actual window used for reporting can be modified by active grabs, and, in the case of keyboard events, by the focus window.

The *root* is the root window of the source window.

The *root-x* and *root-y* are the pointer coordinates relative to the origin of the *root* at the time of the event.

The *event* window is the window that receives the the event. If the event window is on the same screen as *root*, then *event-x* and *event-y* are the pointer coordinates relative to the origin of the event window; otherwise, *event-x* and *event-y* are zero.

If the source window is an inferior of the *event window*, then the *child* is set to the child of the event window that is the source window or an ancestor of it. Otherwise, *child* is **None**.

The *detail* component varies with the event type. If the event is:

- A **KeyPress** event or a **KeyRelease**, the component is **KEYCODE**.
- A **ButtonPress** event or a **ButtonRelease** event, the component is **BUTTON**.
- A **MotionNotify** event, the component is **Normal** or **Hint**.

MotionNotify events are generated when the motion begins and ends in the window. The granularity of motion events is not guaranteed, but a client requesting motion events will receive at least one event when the pointer moves and comes to rest.

By selecting **PointerMotion**, events are received independent of the state of the pointer buttons. By selecting some subset of **Button[1-5]Motion** instead, **MotionNotify** events are received only when one or more of the specified buttons are pressed.

By selecting **ButtonMotion**, **MotionNotify** events are received only when at least one button is pressed. The events are always of type **MotionNotify**,

¦ Copyright IBM Corp. 1985, 1991

independent of the selection. If **PointerMotionHint** is selected, the server is free to send only one **MotionNotify** event (with *detail* **Hint**) to the client for the event window, until the key or button state changes, or the pointer leaves the event window, or the client issues a **QueryPointer** or **GetMotionEvents** request.

The *state* component gives the state of the buttons and modifier keys just prior to the event.

¦ Copyright IBM Corp. 1985, 1991

5.14.16 MapNotify

event, window: **WINDOW**
override-redirect: **BOOL**

MapNotify is reported to clients selecting **StructureNotify** on the window and to clients selecting **SubstructureNotify** on the parent. It is generated when the window changes from an unmapped state to a mapped state.

The *event* is the window on which the event is generated.

The *window* is the window that is mapped.

The *override-redirect* is taken from the window attribute.

¦ Copyright IBM Corp. 1985, 1991

5.14.17 MappingNotify

request: {**Modifier, Keyboard, Pointer**}
first-keycode, count: **CARD8**

MappingNotify event is sent to all clients. No mechanism exists to express disinterest in this event.

The *request* indicates the kind of change that occurred. It can be one of the following:

Modifiers, if **SetModifierMapping** is successful.

Keyboard, if **ChangeKeyboardMapping** is successful. In this case, the *first-keycode* and *count* indicate the range of altered keycodes.

Pointer, if **SetPointerMapping** is successful.

¦ Copyright IBM Corp. 1985, 1991

5.14.18 MapRequest

parent, window: **WINDOW**

MapRequest event is reported to the client selecting **SubstructureRedirect** on the parent. It is generated when a **MapWindow** request is issued on an unmapped window with the *override-redirect* set to **False**.

¦ Copyright IBM Corp. 1985, 1991

5.14.19 NoExposure

drawable: **DRAWABLE**
major-opcode: **CARD8**
minor-opcode: **CARD16**

NoExposure event is reported to clients selecting graphics-exposures in a graphics context. It is generated when a graphics request that should produce **GraphicsExposure** events does not produce any graphics request.

The *drawable* specifies the destination for the graphics request.

The *major-opcode* and *minor opcode* identify the graphics request. For the Core protocol, *major-opcode* is **CopyArea** or **CopyPlane** and *minor-opcode* is always zero.

¦ Copyright IBM Corp. 1985, 1991

5.14.20 PropertyNotify

window: **WINDOW**
atom: **ATOM**
state: {**NewValue, Deleted**}
time: **TIMESTAMP**

PropertyNotify event is reported to clients selecting **PropertyChange** on the window. It is generated when a property of the window is changed by one of the following requests:

ChangeProperty
DeleteProperty
RotateProperties
GetProperty.

The **TIMESTAMP** indicates the server time when the property was changed.

¦ Copyright IBM Corp. 1985, 1991

5.14.21 ReparentNotify

event, window, parent: **WINDOW**
x, y: **INT16**
override-redirect: **BOOL**

ReparentNotify event is reported to clients selecting **SubstructureNotify** on the old or the new *parent* window and to clients selecting **StructureNotify** on the window. It is generated when the window is reparented.

The *event* is the window on which the event was generated.

The *window* is the window that has been re-rooted.

The *parent* specifies the new parent.

The *x* and *y* coordinates, which are relative to the origin of the new *parent*, specify the upper-left outer corner of the window.

The *override-redirect* is taken from the attribute of the window.

¦ Copyright IBM Corp. 1985, 1991

5.14.22 ResizeRequest

window: **WINDOW**
width, height: **CARD16**

ResizeRequest event is reported to the client selecting **ResizeRedirect** on the window. It is generated when a **ConfigureWindow** request by another client on the window attempts to change the size of the window.

The *width* and *height* are the inside of the window, excluding the border.

¦ Copyright IBM Corp. 1985, 1991

5.14.23 SelectionClear

owner: **WINDOW**
selection: **ATOM**
time:**TIMESTAMP**

SelectionClear event is reported to the current *owner* of a *selection*. It is generated when a new owner is being defined by **SetSelectionOwner**.

The *time* is the last-change time recorded for the *selection*.

The *owner* is the window specified by the current *owner* in the **SetSelectionOwner** request.

¦ Copyright IBM Corp. 1985, 1991

5.14.24 SelectionNotify

requestor: **WINDOW**
selection, target: **ATOM**
property: **ATOM** or **None**
time:**TIMESTAMP** or **CurrentTime**

SelectionNotify event is generated by the server in response to a **ConvertSelection** request when there is no owner for the selection. If there is an owner, this event should be generated with **SendEvent**.

The *owner* of a *selection* should send this event to a *requestor* when a *selection* has been converted and stored as a *property* or when a *selection* conversion could not be performed. If *property* is **None**, *selection* is not performed.

¦ Copyright IBM Corp. 1985, 1991

5.14.25 SelectionRequest

owner: **WINDOW**
selection: **ATOM**
target: **ATOM**
property: **ATOM** or **None**
requestor: **WINDOW**
time:**TIMESTAMP** or **CurrentTime**

SelectionRequest event is reported to the *owner* of a *selection*. It is generated when a client issues a **ConvertSelection** request.

The *owner* is the window specified in the **SetSelectionOwner** request. The remaining arguments are the same as in the **ConvertSelection** request.

The *owner* should convert the *selection* based on the specified *target* type.

If a *property* is specified, the *owner* should store the result as that property on the *requestor* window and send a **SelectionNotify** event to the *requestor* using **SendEvent** with an empty *event-mask*. The event should be sent to the creator of the *requestor* window.

If the *selection* cannot be converted as requested, the *owner* should send a **SelectionNotify** with the property set to **None**.

¦ Copyright IBM Corp. 1985, 1991

5.14.26 UnmapNotify

event, window: **WINDOW**
from-configure: **BOOL**

UnmapNotify event is reported to clients selecting **StructureNotify** on the window and to clients selecting **SubstructureNotify** on the parent. It is generated when the window changes from a mapped state to an unmapped state.

The *event* is the window on which the event is generated.

The *window* is the window that is unmapped.

The *from-configure* is **True** if the event is generated as a result of the parent window being resized when the window had a *win-gravity* of **Unmap**.

¦ Copyright IBM Corp. 1985, 1991

5.14.27 VisibilityNotify

window: **WINDOW**
state: {**Unobscured, PartiallyObscured, FullyObscured**}

VisibilityNotify event is reported to clients selecting **VisibilityChange** on the window. (These events are not generated on **InputOnly** windows.)

The *state* of the window is calculated without the subwindows. If *state* is:

Unobscured, a window changed state from partially or fully obscured or not viewable to viewable and completely unobscured.

PartiallyObscured, a window changed state from viewable and completely unobscured or not viewable to viewable and partially obscured.

FullyObscured, a window changed state from viewable and completely unobscured or viewable and partially obscured or not viewable, to viewable and fully obscured.

VisibilityNotify events caused by a hierarchy change are generated after the hierarchy event (**UnmapNotify, MapNotify, ConfigureNotify, GravityNotify**, and **CirculateNotify**) that caused the change.

VisibilityNotify events on a window are generated before **Expose** events on that window, but not all **VisibilityNotify** events on all windows must be generated before all **Expose** events on all windows. The order of **VisibilityNotify** events with respect to **FocusOut**, **EnterNotify**, and **LeaveNotify** events is not constrained.

¦ Copyright IBM Corp. 1985, 1991

5.15 Xlib Functions and Protocol Requests

The following tables provide information about **Xlib** and the X-Windows protocol requests. The first section lists **Xlib** functions alphabetically with the corresponding protocol request that it generates. The second section lists X-Windows protocol requests alphabetically and the **Xlib** functions that reference it.

Xlib Function	**Protocol Request**
XActivateScreenSaver	ForceScreenSaver
XAddHost	ChangeHosts
XAddHosts	ChangeHosts
XAddToSaveSet	ChangeSaveSet
XAllocColor	AllocColor
XAllocColorCells	AllocColorCells
XAllocColorPlanes	AllocColorPlanes
XAllocNamedColor	AllocNamedColor
XAllowEvents	AllowEvents
XAutoRepeatOff	ChangeKeyboardControl
XAutoRepeatOn	ChangeKeyboardControl
XBell	Bell
XChangeActivePointerGrab	ChangeActivePointerGrab
XChangeGC	ChangeGC
XChangeKeyboardControl	ChangeKeyboardControl
XChangeKeyboardMapping	ChangeKeyboardMapping
XChangePointerControl	ChangePointerControl
XChangeProperty	ChangeProperty
XChangeSaveSet	ChangeSaveSet
XChangeWindowAttributes	ChangeWindowAttributes
XCirculateSubwindows	CirculateWindow
XCirculateSubwindowsDown	CirculateWindow
XCirculateSubwindowsUp	CirculateWindow
XClearArea	ClearArea
XClearWindow	ClearArea

¦ Copyright IBM Corp. 1985, 1991

Xlib Function	**Protocol Request**
XConfigureWindow	ConfigureWindow
XConvertSelection	ConvertSelection
XCopyArea	CopyArea
XCopyColormapAndFree	CopyColormapAndFree
XCopyGC	CopyGC
XCopyPlane	CopyPlane
XCreateColormap	CreateColormap
XCreateFontCursor	CreateGlyphCursor
XCreateGC	CreateGC
XCreateGlyphCursor	CreateGlyphCursor
XCreatePixmap	CreatePixmap
XCreatePixmapCursor	CreateCursor
XCreateSimpleWindow	CreateWindow
XCreateWindow	CreateWindow
XDefineCursor	ChangeWindowAttributes
XDeleteProperty	DeleteProperty
XDestroySubwindows	DestroySubwindows
XDestroyWindow	DestroyWindow
XDisableAccessControl	SetAccessControl
XDrawArc	PolyArc
XDrawArcs	PolyArc
XDrawImageString	ImageText8
XDrawImageString16	ImageText16
XDrawLine	PolySegment
XDrawLines	PolyLine
XDrawPoint	PolyPoint
XDrawPoints	PolyPoint
XDrawRectangle	PolyRectangle
XDrawRectangles	PolyRectangle

¦ Copyright IBM Corp. 1985, 1991

XDrawSegments	PolySegment
XDrawString	PolyText8
XDrawString16	PolyText16
XDrawText	PolyText8
XDrawText16	PolyText16
XEnableAccessControl	SetAccessControl
XFetchBytes	GetProperty
XFetchName	GetProperty
XFillArc	PolyFillArc
XFillArcs	PolyFillArc
XFillPolygon	FillPoly
XFillRectangle	PolyFillRectangle
XFillRectangles	PolyFillRectangle
Xlib Function	**Protocol Request**
XForceScreenSaver	ForceScreenSaver
XFreeColormap	FreeColormap
XFreeColors	FreeColors
XFreeCursor	FreeCursor
XFreeFont	CloseFont
XFreeGC	FreeGC
XFreePixmap	FreePixmap
XGetAtomName	GetAtomName
XGetFontPath	GetFontPath
XGetGeometry	GetGeometry
XGetIconSizes	GetProperty
XGetImage	GetImage
XGetInputFocus	GetInputFocus
XGetKeyboardContol	GetKeyboardControl
XGetKeyboardMapping	GetKeyboardMapping
XGetMotionEvents	GetMotionEvents

¦ Copyright IBM Corp. 1985, 1991

XGetNormalHints	GetProperty
XGetPointerContol	GetPointerControl
XGetPonterMapping	GetPointerMapping
XGetScreenSaver	SetScreenSaver
XGetSelectionOwner	GetSelectionOwner
XGetSizeHints	GetProperty
XGetWMHints	GetProperty
XGetWindowAttributes	GetWindowAttributes
XGetWindowAttributes	GetGeometry
XGetWindowProperty	GetProperty
XGetZoomHints	GetProperty
XGrabButton	GrabButton
XGrabKey	GrabKey
XGrabKeyboard	GrabKeyboard
XGrabPointer	GrabPointer
XGrabServer	GrabServer
XInitExtension	QueryExtension
XInstallColormap	InstallColormap
XInternAtom	InternAtom
XKillClient	KillClient
Xlib Function	**Protocol Request**
XListExtensions	ListExtensions
XListFonts	ListFonts
XListFontsWithInfo	ListFontsWithInfo
XListHosts	ListHosts
XListInstalledColormaps	ListInstalledColormaps
XListProperties	ListProperties
XLoadFont	OpenFont
XLoadQueryFont	OpenFont
	QueryFont

¦ Copyright IBM Corp. 1985, 1991

XLookupColor	LookupColor
XLowerWindow	ConfigureWindow
XMapRaised	ConfigureWindow
	MapWindow
XMapSubwindows	MapSubwindows
XMapWindow	MapWindow
XMoveResizeWindow	ConfigureWindow
XMoveWindow	ConfigureWindow
XNoOp	NoOperation
XOpenDisplay	CreateGC
XParseColor	LookupColor
XPutImage	PutImage
XQueryBestCursor	QueryBestSize
XQueryBestSize	QueryBestSize
XQueryBestStipple	QueryBestSize
XQueryBestTile	QueryBestSize
XQueryColor	QueryColors
XQueryColors	QueryColors
XQueryExtension	QueryExtension
XQueryKeymap	QueryKeymap
XQueryPointer	QueryPointer
XQueryTextExtents	QueryTextExtents
XQueryTextExtents16	QueryTextExtents
XQueryTree	QueryTree
XRaiseWindow	ConfigureWindow
XRecolorCursor	RecolorCursor
XRemoveFromSaveSet	ChangeSaveSet
Xlib Function	**Protocol Request**
XRemoveHost	ChangeHosts
XRemoveHosts	ChangeHosts

¦ Copyright IBM Corp. 1985, 1991

XReparentWindow	ReparentWindow
XResetScreenSaver	ForceScreenSaver
XResizeWindow	ConfigureWindow
XRestackWindows	ConfigureWindow
XRotateBuffers	RotateProperties
XRotateWindowProperties	RotateProperties
XSelectInput	ChangeWindowAttributes
XSendEvent	SendEvent
XSetAccessControl	SetAccessControl
XSetArcMode	ChangeGC
XSetBackground	ChangeGC
XSetClipMask	ChangeGC
XSetClipOrigin	ChangeGC
XSetClipRectangles	SetClipRectangles
XSetCloseDownMode	SetCloseDownMode
XSetCommand	ChangeProperty
XSetDashes	SetDashes
XSetFillRule	ChangeGC
XSetFillStyle	ChangeGC
XSetFont	ChangeGC
XSetFontPath	SetFontPath
XSetForeground	ChangeGC
XSetFunction	ChangeGC
XSetGraphicsExposures	ChangeGC
XSetIconSizes	ChangeProperty
XSetInputFocus	SetInputFocus
XSetLineAttributes	ChangeGC
XSetModifierMapping	SetModifierMapping
XSetNormalHints	ChangeProperty
XSetPlaneMask	ChangeGC

¦ Copyright IBM Corp. 1985, 1991

XSetPointerMapping	SetPointerMapping
XSetScreenSaver	SetScreenSaver
XSetSelectionOwner	SetSelectionOwner
XSetSizeHints	ChangeProperty
Xlib Function	**Protocol Request**
XSetStandardProperties	ChangeProperty
XSetState	ChangeGC
XSetStipple	ChangeGC
XSetSubwindowMode	ChangeGC
XSetTile	ChangeGC
XSetTSOrigin	ChangeGC
XSetWMHints	ChangeProperty
XSetWindowBackground	ChangeWindowAttributes
XSetWindowBackgroundPixmap	ChangeWindowAttributes
XSetWindowBorder	ChangeWindowAttributes
XSetWindowBorderPixmap	ChangeWindowAttributes
XSetWindowBorderWidth	ConfigureWindow
XSetWindowColormap	ChangeWindowAttributes
XSetZoomHints	ChangeProperty
XStoreBuffer	ChangeProperty
XStoreBytes	ChangeProperty
XStoreColor	StoreColors
XStoreColors	StoreColors
XStoreName	ChangeProperty
XStoreNamedColor	StoreNamedColor
XSync	GetInputFocus
XTranslateCoordinates	TranslateCoordinates
XUndefineCursor	ChangeWindowAttributes
XUngrabButton	UngrabButton
XUngrabKey	UngrabKey

¦ Copyright IBM Corp. 1985, 1991

XUngrabKeyboard	UngrabKeyboard
XUngrabPointer	UngrabPointer
XUngrabServer	UngrabServer
XUninstallColormap	UninstallColormap
XUnloadFont	CloseFont
XUnmapSubwindows	UnmapSubwindows
XUnmapWindow	UnmapWindow
XWarpPointer	WarpPointer

Protocol Request	**Xlib Function**
AllocColor	**XAllocColor**
AllocColorCells	**XAllocColorCells**
AllocColorPlanes	**XAllocColorPlanes**
AllocNamedColor	**XAllocNamedColor**
AllowEvents	**XAllowEvents**
Bell	**XBell**
SetAccessControl	**XDisableAccessControl**
	XEnableAccessControl
	XSetAccessControl
ChangeActivePointerGrab	**XChangeActivePointerGrab**
SetCloseDownMode	**XSetCloseDownMode**
ChangeGC	**XChangeGC**
	XSetArcMode
	XSetBackground
	XSetClipMask
	XSetClipOrigin
	XSetFillRule
	XSetFillStyle
	XSetFont
	XSetForeground
	XSetFunction

¦ Copyright IBM Corp. 1985, 1991

	XSetGraphicsExposures
	XSetLineAttributes
	XSetPlaneMask
	XSetState
	XSetStipple
	XSetSubwindowMode
	XSetTile
	XSetTSOrigin
ChangeHosts	**XAddHost**
	XAddHosts
	XRemoveHost
	XRemoveHosts
ChangeKeyboardControl	**XAutoRepeatOff**
	XAutoRepeatOn
	XChangeKeyboardControl
Protocol Request	**Xlib Function**
ChangeKeyboardMapping	**XChangeKeyboardMapping**
ChangePointerControl	**XChangePointerControl**
ChangeProperty	**XChangeProperty**
	XSetCommand
	XSetIconSizes
	XSetNormalHints
	XSetSizeHints
	XSetStandardProperties
	XSetWMHints
	XSetZoomHints
	XStoreBuffer
	XStoreBytes
	XStoreName
ChangeSaveSet	**XAddToSaveSet**

¦ Copyright IBM Corp. 1985, 1991

	XChangeSaveSet
	XRemoveFromSaveSet
ChangeWindowAttributes	**XChangeWindowAttributes**
	XDefineCursor
	XSelectInput
	XSetWindowBackground
	XSetWindowBackgroundPixmap
	XSetWindowBorder
	XSetWindowBorderPixmap
	XSetWindowColormap
	XUndefineCursor
CirculateWindow	**XCirculateSubwindowsDown**
	XCirculateSubwindowsUp
	XCirculateSubwindows
ClearArea	**XClearArea**
	XClearWindow
CloseFont	**XFreeFont**
	XUnloadFont
Protocol Request	**Xlib Function**
ConfigureWindow	**XConfigureWindow**
	XLowerWindow
	XMapRaised
	XMoveResizeWindow
	XMoveWindow
	XRaiseWindow
	XResizeWindow
	XRestackWindows
	XSetWindowBorderWidth
ConvertSelection	**XConvertSelection**
CopyArea	**XCopyArea**

¦ Copyright IBM Corp. 1985, 1991

CopyColormapAndFree	**XCopyColormapAndFree**
CopyGC	**XCopyGC**
CopyPlane	**XCopyPlane**
CreateColormap	**XCreateColormap**
CreateCursor	**XCreatePixmapCursor**
CreateGC	**XCreateGC**
	XOpenDisplay
CreateGlyphCursor	**XCreateFontCursor**
	XCreateGlyphCursor
CreatePixmap	**XCreatePixmap**
CreateWindow	**XCreateSimpleWindow**
	XCreateWindow
DeleteProperty	**XDeleteProperty**
DestroySubwindows	**XDestroySubwindows**
DestroyWindow	**XDestroyWindow**
FillPoly	**XFillPolygon**
ForceScreenSaver	**XActivateScreenSaver**
	XForceScreenSaver
	XResetScreenSaver
FreeColormap	**XFreeColormap**
FreeColors	**XFreeColors**
FreeCursor	**XFreeCursor**
FreeGC	**XFreeGC**
FreePixmap	**XFreePixmap**
GetAtomName	**XGetAtomName**
Protocol Request	**Xlib Function**
GetFontPath	**XGetFontPath**
GetGeometry	**XGetGeometry**
	XGetWindowAttributes
GetImage	**XGetImage**

¦ Copyright IBM Corp. 1985, 1991

GetInputFocus	**XGetInputFocus**
	XSync
GetKeyboardControl	**XGetKeyboardContol**
GetKeyboardMapping	**XGetKeyboardMapping**
GetMotionEvents	**XGetMotionEvents**
GetPointerControl	**XGetPointerContol**
GetPointerMapping	**XGetPonterMapping**
GetProperty	**XFetchBytes**
	XFetchName
	XGetIconSizes
	XGetNormalHints
	XGetSizeHints
	XGetWMHints
	XGetWindowProperty
	XGetZoomHints
GetSelectionOwner	**XGetSelectionOwner**
GetWindowAttributes	**XGetWindowAttributes**
GrabButton	**XGrabButton**
GrabKey	**XGrabKey**
GrabKeyboard	**XGrabKeyboard**
GrabPointer	**XGrabPointer**
GrabServer	**XGrabServer**
ImageText16	**XDrawImageString16**
ImageText8	**XDrawImageString**
InstallColormap	**XInstallColormap**
InternAtom	**XInternAtom**
KillClient	**XKillClient**
ListExtensions	**XListExtensions**
ListFonts	**XListFonts**
ListFontsWithInfo	**XListFontsWithInfo**

¦ Copyright IBM Corp. 1985, 1991

ListHosts	**XListHosts**
ListInstalledColormaps	**XListInstalledColormaps**
Protocol Request	**Xlib Function**
ListProperties	**XListProperties**
LookupColor	**XLookupColor**
	XParseColor
MapSubwindows	**XMapSubwindows**
MapWindow	**XMapRaised**
	XMapWindow
NoOperation	**XNoOp**
OpenFont	**XLoadFont**
	XLoadQueryFont
PolyArc	**XDrawArc**
	XDrawArcs
PolyFillArc	**XFillArc**
	XFillArcs
PolyFillRectangle	**XFillRectangle**
	XFillRectangles
PolyLine	**XDrawLines**
PolyPoint	**XDrawPoint**
	XDrawPoints
PolyRectangle	**XDrawRectangle**
	XDrawRectangles
PolySegment	**XDrawLine**
	XDrawSegments
PolyText16	**XDrawString16**
	XDrawText16
PolyText8	**XDrawString**
	XDrawText
PutImage	**XPutImage**

¦ Copyright IBM Corp. 1985, 1991

QueryBestSize	**XQueryBestCursor**
	XQueryBestSize
	XQueryBestStipple
	XQueryBestTile
QueryColors	**XQueryColor**
	XQueryColors
QueryExtension	**XInitExtension**
	XQueryExtension
QueryFont	**XLoadQueryFont**
Protocol Request	**Xlib Function**
QueryKeymap	**XQueryKeymap**
QueryPointer	**XQueryPointer**
QueryTextExtents	**XQueryTextExtents**
	XQueryTextExtents16
QueryTree	**XQueryTree**
RecolorCursor	**XRecolorCursor**
ReparentWindow	**XReparentWindow**
RotateProperties	**XRotateBuffers**
	XRotateWindowProperties
SendEvent	**XSendEvent**
SetClipRectangles	**XSetClipRectangles**
SetCloseDownMode	**XSetCloseDownMode**
SetDashes	**XSetDashes**
SetFontPath	**XSetFontPath**
SetInputFocus	**XSetInputFocus**
SetModifierMapping	**XSetModifierMapping**
SetPointerMapping	**XSetPointerMapping**
SetScreenSaver	**XGetScreenSaver**
	XSetScreenSaver
SetSelectionOwner	**XSetSelectionOwner**

¦ Copyright IBM Corp. 1985, 1991

StoreColors	**XStoreColor**
	XStoreColors
StoreNamedColor	**XStoreNamedColor**
TranslateCoordinates	**XTranslateCoordinates**
UngrabButton	**XUngrabButton**
UngrabKey	**XUngrabKey**
UngrabKeyboard	**XUngrabKeyboard**
UngrabPointer	**XUngrabPointer**
UngrabServer	**XUngrabServer**
UninstallColormap	**XUninstallColormap**
UnmapSubwindows	**XUnmapSubWindows**
UnmapWindow	**XUnmapWindow**
WarpPointer	**XWarpPointer**

¦ Copyright IBM Corp. 1985, 1991

6.0 Chapter 6. Extensions

Subtopics

¦ Copyright IBM Corp. 1985, 1991

6.1 CONTENTS

¦ Copyright IBM Corp. 1985, 1991

6.2 About This Chapter

This chapter describes techniques for writing extensions to **Xlib** that have the same performance rate as Core protocol requests. "Using AIX Extensions" in topic 6.13 describes AIX X-Windows extensions, which are supported generally on the RT only.

Note: Because an X-Windows extension is expected to consist of multiple requests, defining 10 new features as 10 separate extensions is not a good practice. Rather, package new features into a single extension and use minor opcodes to distinguish between the features.

¦ Copyright IBM Corp. 1985, 1991

6.3 Basic Extension Routines

The basic protocol requests for extensions are **XQueryExtension**, **XListExtensions**, and **XFreeExtensionList**. These and other extensions are described in this chapter.

¦ Copyright IBM Corp. 1985, 1991

6.4 Hooking into Xlib

Hooking routines sink a connecting hook the library. These routines normally are not used by application programmers but, instead, by programmers who need to extend the Core X-Windows protocol and the X-Windows library interface. Hooking routines, which generate protocol requests for X-Windows, are called ***stubs***.

In extensions, stubs first check to see if they have initialized themselves on a connection. If the stubs have not been initialized, they should call **XInitExtension**.

The wire-formatted structure **xEvent** is in **<X11/Xproto.h>** and the host-formatted structure **XEvent** is in **<X11/Xlib>**.

The following **_XExtCodes** structure returns the information from **XQueryExtension**. This structure is public to extension and cannot be changed.

```
typedef struct _XExtCodes {

  int extension;                /* extension number                  */
  int major_opcode;             /* major opcode assigned by server   */
  int first_event;              /* first event number for the        */
                                   extension
  int first_error;              /* first error number for the        */
                                   extension

} XExtCodes;
```

```
XExtCodes *XInitExtension(display, name)
      Display *display;
      char *name;
```

XInitExtension calls **XQueryExtension** to see if the extension exists. Then, it allocates storage for maintaining the information about the extension on the connection. It chains this to the extension list for the connection, and returns the information the stub implementor needs to access the extension.

The extension number returned in the **XExtCodes** structure is used in other calls. This extension number is unique to a single connection only.

Subtopics
6.4.1 Hooks into the Library

¦ Copyright IBM Corp. 1985, 1991

6.4.1 Hooks into the Library

Types of functions and associated routines that hook into the X-Windows library are:

- Creating a new **GC** for a connection (**XESetCloseDisplay** and **XESetCreateGC**)
- Copying a **GC** (**XESetCopyGC** and **XESetFlushGC**)
- Freeing a **GC** (**XESetFreeGC**)
- Creating and freeing fonts **XESetCreateFont** and **XESetFreeFont**)
- Converting events defined by extensions to and from wire forma (**XESetWireToEvent** and **XESetEventToWire**)
- Handling errors **XESetError**, **XESetErrorString**, and **XGetErrorText**).

Use these routines to define procedures to be called under certain circumstances. All these routines return the previous routine defined for this extension.

¦ Copyright IBM Corp. 1985, 1991

6.5 GC Caching

GCs are cached by the library so that independent change requests can be merged into a single protocol request. This cache is called a ***write back cache***. Any extension routine whose behavior depends on the contents of a **GC** must flush the **GC** cache to make sure the server has up-to-date contents in its **GC**.

If you extend the **GC** to add additional resource ID components, you should ensure that the library stub immediately sends the change request. Since a client can free a resource immediately after using it, storing the value in the cache without forcing a protocol request can destroy the resource before it is set into the **GC**.

The **_XFlushGCCache** procedure forces the cache to be flushed.

¦ Copyright IBM Corp. 1985, 1991

6.6 Graphics Batching

If you extend X-Windows to add more poly-graphics primitives, you might be able to take advantage of facilities in the library to allow back-to-back single calls to be transformed into poly-requests. The display structure has a pointer to an **xReq** called **last_req**, which is the last request being processed. By checking that the last request type, drawable, **GC**, and other options are the same as the new one, and that there is enough space left in the buffer, you might be able to extend the previous graphics request by extending the length field of the request and appending the data to the buffer.

For example, here is the source for the **XDrawPoint** stub:

```
#include <X11/Xlibint.h>

/* precompute the max size of batching request allowed */

    static int size = sizeof(xPolyPointReq) + EPERBATCH * sizeof(xPoint);

XDrawPoint(dpy, drawable, gc, x, y)
    register Display *dpy;
    Drawable drawable;
    GC gc;
    int x, y;                   /* INT16 */

{

    xPoint *point;
    LockDisplay(display);
    FlushGC(display, gc);

    {

    register xPolyPointReq *req = (xPolyPointReq *) display->last_req;

    /* if same as previous request, with same drawable, batch requests */

    if (
          (req->reqType == X_PolyPoint)
       && (req->drawable == d)
       && (req->gc == gc->gid)
       && (req->coordMode == CoordModeOrigin)
       && ((display->bufptr + sizeof(xPoint)) <=display->bufmax)
       && (((char *)display->bufptr - (char *)req) < size) ) {
         point = (xPoint *) display->bufptr;
         req->length += sizeof (xPoint) >> 2;
         display->bufptr += sizeof (xPoint);
         }

    else {
        GetReqExtra(PolyPoint, 4, req); /* 1 point = 4 bytes */
        req->drawable = drawable;
        req->gc = gc->gid;
        req->coordMode = CoordModeOrigin;
        point = (xPoint *) (req + 1);
        }
    point->x = x;
    point->y = y;
    }
```

¦ Copyright IBM Corp. 1985, 1991

```
    UnlockDisplay(display);
    SyncHandle();
}
```

To keep clients from generating long requests that might monopolize the server, there is a limit of **EPERBATCH** on the number of requests batched. Note that **FlushGC** is called before picking up the value of *last_req*, since it may modify this field.

¦ Copyright IBM Corp. 1985, 1991

6.7 Writing Extension Stubs

X Server requests contain the length, expressed in 16-bit quantity of 32-bits, of the request. Therefore, a single request can be no more than 256 kilobytes in length. Some servers may not support single requests of such a length. The value of *display->max_request_size* contains the maximum length as defined by the server implementation.

¦ Copyright IBM Corp. 1985, 1991

6.8 Defining Requests and Replies

The **<X11/Xproto.h>** header file contains three sets of definitions:

- Request name
- Request structure
- Reply structures

Generate a file equivalent to **<X11/Xproto.h>** for your extension and include it in your stub routine. Each stub routine also must include **<X11/Xlibint.h>**.

The identifiers are deliberately chosen in such a way that if the request is called **X_DoSomething**, then its request structure is **xDoSomethingReq** and its reply is **xDoSomethingReply**. The **GetReq** family of macros, defined in **<X11/Xlibint.h>**, takes advantage of this naming scheme.

For each X Request, there is a definition in **<X11/Xproto.h>** that looks similar to the following:

```
#define X_DoSomething   42
```

In your extension header file, this is a minor opcode instead of a major opcode.

Subtopics

¦ Copyright IBM Corp. 1985, 1991

6.8.1 Defining Request Format

Every request contains an 8-bit major opcode and a 16-bit length field expressed in units of 4 bytes. Every request consists of a 4-byte header (containing the major opcode, the length field, and a data byte) followed by a zero or additional bytes of data. The length field defines the total length of the request, including the header. The length field in a request must equal the minimum length required to contain the request. If the specified length is smaller or larger than the required length, the extension should generate a **BadLength** error. Unused bytes in a request are not required to be zero.

Major opcodes 128 through 255 are reserved for extensions. Extensions are for holding multiple requests, therefore extension requests typically have an additional minor opcode encoded in the spare data byte in the request header. But the placement and interpretation of this minor opcode as well as all other fields in extension requests are not defined by the Core protocol. Every request is implicitly assigned a sequence number (starting with one) used in replies, errors, and events.

Most protocol requests have a corresponding structure **typedef** in **<X11/Xproto.h>**. This is an example of a **typedef** structure:

```
typedef struct _DoSomethingReq {

  CARD8 reqType;              /* X_DoSomething                    */
  CARD8 someDatum;            /* used differently in different    */
                              requests
  CARD16 length;              /* total number of bytes in         */
                              request, divided by 4
  ....                        /* request-specific data            */
  ...

} xDoSomethingReq;
```

If a Core protocol request has a single 32-bit argument, you do not need to declare a request structure in your extension header file. Instead, such requests use the **xResourceReq** structure in **<X11/Xproto.h>**. This structure is used for any request whose single argument is *window, pixmap, drawable, GContext, font, cursor, colormap, device, atom, visualID,* or *time*.

The following is an example of the **xResourceReq** typedef structure:

```
typedef struct _ResourceReq {

  CARD8 reqType;              /* the request type, X_DoSomething  */
  BYTE pad;                   /* not used                         */
  CARD16 length;              /* 2 (= total number of bytes in    */
                              request, divided by 4)
  CARD32 id;                  /* the window, drawable, font, or   */
                              gcontext, for example

} xResourceReq;
```

You can do something similar in your extension header file. In both structures, the *reqType* field identifies the type of the request, such as

¦ Copyright IBM Corp. 1985, 1991

X_MapWindow or **X_CreatePixmap**.

The *length* tells how long (in units of 4 bytes) the request is. It includes both the request structure and any variable length data, such as strings or lists, that follow the request structure. Request structures come in different sizes, but all requests are padded to be a multiple of 4-bytes long.

A few protocol requests take no arguments at all. Instead, they use the **xReq** structure, which contains only a *reqType* and a length (and a pad byte), in **<X11/Xproto.h>**.

If the protocol request requires a reply, then **<Xproto.h>** also contains a reply structure **typedef** such as the following:

```
typedef struct _DoSomethingReply {

  BYTE type;                    /* always X_Reply                  */
  BYTE someDatum;               /* used differently in different   */
                                   requests
  CARD16 sequenceNumber;        /* number of requests sent so      */
                                   far
  CARD32 length;                /* number of additional bytes,     */
                                   divided by 4
  ....
                                /* request-specific data           */
  ....

} xDoSomethingReply;
```

Most of these reply structures are 32 bytes long. If the reply value is less than 32 bytes, the reply structure contains sufficient number of pad fields to bring them up to 32 bytes.

The *length* is the total number of bytes in the request minus 32, divided by 4. This field is not zero if:

The reply structure is followed by variable length data such as a lis or string

The reply structure is longer than 32 bytes

Only **GetWindowAttributes, QueryFont, QueryKeymap**, and **GetKeyboardControl** have reply structures longer than 32 bytes.

A few protocol requests return replies that contain no data. **<X11/Xproto.h>** does not define reply structures for these. Instead, these protocol requests use the **xGenericReply** structure, which contains only a type, length, and sequence number (and sufficient padding to make it 32-bytes long).

¦ Copyright IBM Corp. 1985, 1991

6.8.2 Writing a Stub Routine

An **Xlib** stub routine should start as follows:

```
#include <X11/Xlibint.h>

        XDoSomething (arguments, ...)     /* argument declarations      */
        {
                                          /* variable declarations,     */
                                          if any
```

If the protocol request has a reply, then the variable declarations should include the reply structure for the request. The following is an example of a stub routine:

```
xDoSomethingReply rep;
```

¦ Copyright IBM Corp. 1985, 1991

6.8.3 Locking Data Structures

To support asynchronous input, the display must be locked so that each stub can lock its critical section. Generally, this section is the point immediately prior to the appropriate **GetReq** call when all arguments to the call have been stored into the request. Two calls generally implemented as macros are:

LockDisplay(*display*)
   Display *display;

UnlockDisplay:(*display*)
   Display *display;

The *display* variable specifies the display to be locked or unlocked.

¦ Copyright IBM Corp. 1985, 1991

6.8.4 Sending the Protocol Request and Arguments

After the variable declarations, a stub routine should call one of four macros defined in the **Xlibint.h** file:

GetReq
GetReqExtra
GetResReq
GetEmptyReq

These macros take the name of the protocol request as declared in **<X11/Xproto.h>** without the **X_**, as their first argument. Each macro declares a **Display** structure pointer, called *display* and a pointer to a request structure, called *req*, which is of the appropriate type. The macro then appends the request structure to the output buffer, fills in the type and length field, and sets *req* to point to it.

If the protocol request, such as **_GrabServer**, has no arguments, use **GetEmptyReq** as in the following example:

```
GetEmptyReq (DoSomething);
```

If the protocol request has a single 32-bit argument (such as a *pixmap, window, drawable, atom*), use **GetResReq**.

The second argument to this macro is the 32-bit object. **X_MapWindow** is a good example of **GetResReq**:

```
GetResReq (DoSomething, rid);
```

The *rid* argument is the **Pixmap**, **Window**, or other resource ID.

If the protocol request takes any other argument list, then call **GetReq**. After the **GetReq**, set all the other fields in the request structure, usually from arguments to the stub routine.

```
GetReq (DoSomething);

/* fill in arguments here */

req->arg1 = arg1;
req->arg2 = arg2;
```

A few stub routines, such as **XCreateGC** and **XCreatePixmap**, return a resource ID to the caller but pass a resource ID as an argument to the protocol request. These stub routines use the macro **XAllocID** to allocate a resource ID from the range of IDs that were assigned to this client when it opened the connection. The following is an example of **XAllocID**:

```
rid = req->rid = XAllocID();
return (rid);
```

Finally, some stub routines transmit a fixed amount of variable-length data after the request. Typically, these routines, such as **XMoveWindow** and **XSetBackgroundPixel**, are special cases of more general routines like **XMoveResizeWindow** and **XChangeGC**. In these cases, **GetReqExtra**, which is like **GetReq** with an an additional argument, is used. The additional argument is the number of extra bytes (a multiple of 4) allocated in the

¦ Copyright IBM Corp. 1985, 1991

output buffer after the request structure.

¦ Copyright IBM Corp. 1985, 1991

6.8.5 Using Variable Length Arguments

Some protocol requests take additional variable length data that follow the **xDoSomethingReq** structure. The format of this data varies from one request to another. Some require a sequence of 8-bit bytes, others a sequence of 16-bit or 32-bit entities, and still others a sequence of structures.

The length of any variable length data must be added to the length field of the request structure. The length field is in units of 32-bit longwords. If the data is a string or other sequence of 8-bit bytes, then round up the length and shift it before adding. For example:

```
req->length += (nbytes+3)>>2;
```

To transmit the variable length data, use the **Data** macro. If the data fits into the output buffer, then this macro copies it to the buffer. If it does not fit, however, the **Data** macro calls **_XSend**, which first transmits the contents of the buffer and then transmits your data. The **Data** macro takes three arguments:

```
    the displa
    a pointer to the beginning of the dat
    the number of bytes to be sent
```

The following is an example of the **Data** macro:

```
Data(display, (char *) data, nbytes);
```

If the data is 16-bit entities, use the **PackData** macro. It performs correctly on machines where a short is 32-bits instead of the usual 16.

Both **Data** and **PackData** macros can use their last argument more than once, so that *argument* should be a variable rather than an expression, such as **nitems*sizeof(item)**. This sort of computation should be done in a separate statement before calling **Data**.

If the protocol request requires a reply, use **_XSend**. **_XSend** is faster than **Data** macro, because it sends the data immediately instead of copying it into the output buffer.

If the protocol request has a reply, use **_XReply** after dealing with all the fixed and variable length arguments. See "_XReply" in topic 6.12.1.

¦ Copyright IBM Corp. 1985, 1991

6.8.6 Synchronous Calling

To ease debugging, each routine should have a call to a routine immediately prior to returning to the user. This routine is called **SyncHandle()** and is generally implemented as a macro. If **synchronous** mode is enabled, the request is sent immediately. The library, however, waits until any error generated has been handled.

¦ Copyright IBM Corp. 1985, 1991

6.9 Allocating and Deallocating Memory

To support the possible re-entry of these routines, several conventions should be observed when allocating and deallocating memory. This is appropriate especially when the user does not know the size of the data that is being returned. (The standard C language library routines on many systems are not protected against signals or other multi-threaded use.) The analogies to standard I/O library routines are defined as follows:

Xmalloc() Replaces the **malloc()** routine

Xfree() Replaces the **free()** routine

Xcalloc() Replaces the **calloc()** routine.

These routines should be used in place of any calls made to the normal C language library routines. For example, if you need a single scratch buffer inside a critical section to pack and unpack data to and from wire protocol, the general memory allocators may be too expensive to use (particularly in output routines, which are performance critical). Use a routine with the following form to return a scratch buffer:

```
char *_XAllocScratch(display, nbytes)
      Display *display;
      unsigned long nbytes;
```

This storage must only be used inside the critical section of your stub.

¦ Copyright IBM Corp. 1985, 1991

6.10 Deriving the Correct Extension Opcode

When writing an extension stub routine map from the call to the proper major and minor opcodes. While there are a number of strategies, one is outlined here:

1. Declare an array of pointers. The length of this array, **_NFILE long** (normally found in **<stdio.h>**), is the number of file descriptors supported on the system of type **XExtCodes**. These descriptors should be initialized to **NULL**.

2. When your stub is entered, your initialization test should use the display pointer to access the file descriptor and an index into the array. If the entry is **NULL**, then this is the first time you are entering the routine for this display. Call your initialization routine and pass the display pointer.

3. Once in your initialization routine, call **XInitExtension**. If it succeeds, store the pointer returned into this array. Establish a close display handler to allow you to zero the entry. Perform any other initialization your extension requires. (For example, install event handlers.) Your initialization routine normally will return a pointer to the **XExtCodes** structure for this extension, which is should be found in your array of pointers.

4. After the initialization routine, the stub routine can continue normally, since its major opcode is safely in the **XExtCodes** structure.

¦ Copyright IBM Corp. 1985, 1991

6.11 Using Extension Events

This section contains information on:

- Extension event type
- Extension event mask
- Extension event processing

Note: This is supported on the RT only.

Subtopics
6.11.1 Using Event Types
6.11.2 Defining Event Structures
6.11.3 Using Event Masks

¦ Copyright IBM Corp. 1985, 1991

6.11.1 Using Event Types

An extension event is data generated asynchronously by the X Server as a result of some input device activity or as side effects of an extension request. Device-related events propagate from the source window to ancestor windows until some client application has selected that event type or until the event is explicitly discarded. The X Server never sends an event to a client application unless the client has specifically asked to be informed of that event type, usually by calling the **Xlib** function **XSelectDeviceInput**. However, **AIXDeviceMappingNotify** events are always sent.

The event type describes a specific event generated by the X Server. For each event type, a corresponding name is defined in **<X11/AIX.h>**. The following table lists the event category and its associated event type or types.

Event Category	**Event Type**
Lpfk events	**LPFKeyPress**
Dial events	**DialRotate**
Focus and mapping change events	**AIXFocusIn AIXFocusOut AIXDeviceMappingNotify**

¦ Copyright IBM Corp. 1985, 1991

6.11.2 Defining Event Structures

Each event type has a corresponding structure declared in **<X11/AIX.h>**. Event structures have the following fields:

type — Set to the event type constant name that uniquely identifies the type. For example, when the X Server reports a **DialRotate** event to a client application, it sends an **XDialRotatedEvent** structure with the *type* field set to **DialRotate**.

display — Set to a pointer to the display the event was read on.

send_event — Set to **True** if the event came from an **XSendEvent** request.

serial — Set from the serial number reported in the protocol but expanded from the 16-bit least-significant bits to a full 32-bit value.

The X Server can send extension events at any time in the input stream, even while the client application sends a request and receives a reply. Events received, while waiting for a reply, can be stored by **Xlib** in the event queue.

¦ Copyright IBM Corp. 1985, 1991

6.11.3 Using Event Masks

Clients select extension event reporting of most events relative to a window by passing an extension event mask to an **Xlib** event-handling function that takes *ext_event_mask* argument. The bits of the event mask are defined in **<X11/AIX.h>**. Each bit in the event mask maps to an event mask name. The event mask name describes the event or events to be returned to a client application by the server.

The following table lists the event mask that can be specified in the *ext_event_mask* argument and the circumstances under which to specify them.

Event Mask	**Circumstances**
NoEventMask	No events wanted
LPFKeyPressMask	Lpfk key-down events wanted
DialRotateMask	Dail rotate events wanted
AIXFocusChangeMask	Dial or lpfk input focus events wanted
AIXDeviceMapChangeMask	Device state change events wanted

¦ Copyright IBM Corp. 1985, 1991

6.12 Routines

The protocol extension routines are described in the remainder of this chapter. These routines are in alphabetical order.

Subtopics

¦ Copyright IBM Corp. 1985, 1991

6.12.1 _XReply

Status _XReply(*display, rep, extra, discard***)**
Display ***display;**
xReply ***rep;**
int *extra**;**
Bool *discard**;**

display Specifies the display structure.

rep Specifies a pointer to **xReply** structure.

extra Specifies the number of additional bytes (beyond **sizeof(xReply) = 32 bytes**) in the reply structure. This is the number of words expected after the reply.

discard Specifies a Boolean value that tells **_XReply** to discard any additional bytes beyond those it was told to read.

_XReply flushes the output buffer, waits for a reply packet, and copies the contents into the specified *rep*. If other events arrive during this time, **_XReply** queues these events for use later. It handles error and event packets that occur before the reply is received.

Most reply structures are 32 bytes long, therefore, the *extra* is usually zero (0). In the Core protocol, only the following reply structures are longer than 32 bytes: **GetWindowAttributes**, **QueryFont**, **QueryKeymap**, and **GetKeyboardControl**.

The *discard* can be one of the following:

xFalse, if the reply structure is followed by additional variable-length data, such as a list or string.

xTrue, if there is no variable-length data.

If a reply is not followed by variable length data, use **_XReply** as follows:

```
_XReply (display, (xReply *)&rep, 0, xTrue);
*ret1 = rep.ret1;
*ret2 = rep.ret2;
*ret3 = rep.ret3;
UnlockDisplay(dpy);
SyncHandle();
return (rep.ret4);
}
```

If a reply has variable-length data, change the **xTrue** to **xFalse** and use **_XRead** to read the variable-length data.

Note: The *discard* argument is for a client that uses a version of the server that sends more data than the client expects. For example, a later version of **GetWindowAttributes** may use a larger, but compatible **xGetWindowAttributesReply** that contains more data at the end.

In this case, **_XReply** returns one of the following:

True, if a reply was received successfully.

¦ Copyright IBM Corp. 1985, 1991

False and an **XError**, if the reply was not successful.

¦ Copyright IBM Corp. 1985, 1991

6.12.2 XESetCloseDisplay

```
int (*XESetCloseDisplay(display, extension, proc))()
      Display *display;
      int extension;
      int (*proc)();
```

display Specifies the display.

extension Specifies the extension number.

proc Specifies the routine to call when the display is closed.

XESetCloseDisplay defines a procedure to call when **XCloseDisplay** is called. This procedure returns any previously defined procedure, usually **NULL**.

When **XCloseDisplay** is called, the routine is called with these arguments:

```
(*proc)(display, codes)
        Display *display;
        XExtCodes *codes;
```

¦ Copyright IBM Corp. 1985, 1991

6.12.3 XESetCopyGC

```
int (*XESetCopyGC(display, extension, proc))()
      Display *display;
      int extension;
      int (*proc)();
```

display Specifies the display.

extension Specifies the extension number.

proc Specifies the routine to call when a **GC** is copied.

XESetCopyGC defines a procedure to call whenever a GC is copied. This procedure returns any previously defined procedure, usually **NULL**.

When a **GC** is copied, the routine is called with these arguments:

```
(*proc)(display, gc, codes)
        Display *display;
        GC gc;
        XExtCodes *codes;
```

¦ Copyright IBM Corp. 1985, 1991

6.12.4 XESetCreateFont

```
int (*XESetCreateFont(display, extension, proc))()
      Display *display;
      int extension;
      int (*proc)();
```

display Specifies the display.

extension Specifies an extension number.

proc Specifies a routine to call when a font is created.

XESetCreateFont defines a procedure to call when **XLoadQueryFont** is called. This procedure returns any previously defined procedure, usually **NULL**.

When **XLoadQueryFont** is called, the routine is called with these arguments:

```
(*proc)(display, fs, codes)
        Display *display;
        XFontStruct *fs;
        XExtCodes *codes;
```

¦ Copyright IBM Corp. 1985, 1991

6.12.5 XESetCreateGC

```
int (*XESetCreateGC(display, extension, proc))()
      Display *display;
      int extension;
      int (*proc)();
```

display Specifies the display.

extension Specifies the extension number.

proc Specifies the routine to call when a **GC** is created.

XESetCreateGC defines a procedure to call when a new GC is created. It returns any previously defined procedure, usually **NULL**.

When a GC is created, the routine is called with these arguments:

```
(*proc)(display, gc, codes)
        Display *display;
        GC gc;
        XExtCodes *codes;
```

¦ Copyright IBM Corp. 1985, 1991

6.12.6 XESetError

```
int (*XESetError(display, extension, proc))()
      Display *display;
      int extension;
      int (*proc)();
```

display Specifies a display.

extension Specifies an extension number.

proc Specifies a routine to call when an error code is received.

XESetError suppresses the call to an external error handling routine and defines an alternative routine for error handling. It allows status to be returned on a call at the cost of the call being synchronous (though most such routines are query operations and are typically programmed to be synchronous).

When **Xlib** detects a protocol error in **_XReply**, it calls the procedure with these arguments:

```
int (*proc)(display, err, codes, ret_code)
        Display *display;
        xError *err;
        XExtCodes *codes;
        int *ret_code;
```

The *err* argument is a pointer to the 32-byte wire format error.

The *codes* argument is a pointer to the extension codes structure.

The *ret_code* field is the return code returned by **_XReply**.

If the routine returns a zero, the error is not suppressed, and **XError** is called. If the routine returns a non-zero, the error is suppressed and **_XReply** returns the value in *ret_code* argument.

¦ Copyright IBM Corp. 1985, 1991

6.12.7 XESetErrorString

```
char  *(*XESetErrorString(display, extension, proc))()
      Display *display;
      int extension;
      char *(*proc)();
```

display Specifies a display.

extension Specifies an extension number.

proc Specifies a routine to call when an I/O error occurs.

XESetErrorString defines a procedure to call when an I/O error is detected.

¦ Copyright IBM Corp. 1985, 1991

6.12.8 XESetEventToWire

```
int (*XESetEventToWire(display, event_number, proc))()
      Display *display;
      int event_number;
      (*proc)();
```

display Specifies a display.

event_number Specifies a protocol event number to replace with the conversion routine.

proc Specifies a routine to call when converting an event.

XESetEventToWire defines a procedure to call when an event needs to be converted from the host format (**XEvent** found in **<X11/Xlib.h>**) to the wire format (**xEvent** found in **<X11/Xproto.h>**). It returns any previously defined procedure.

Note: The host event structure size cannot be larger than the size of the **XEvent** union of structures.

When **Xlib** needs to convert an event from the wire format to the natural host format, the routine is called with these arguments:

```
(*proc)(display, re, event)
        Display *display;
        XEvent *re;
        xEvent *event;
```

The argument *re* is a pointer to the host format event.

The *event* argument is a pointer to where the 32-byte wire event structure should be stored.

In the **XEvent** structure, the *type* argument should be the first element and the *window* argument should be the second element. The *type* argument should be copied from the **xEvent** structure. The other elements should be copied from the wire format to the **XEvent** structure. For more information, see Chapter 1, "Using X-Windows" in topic 1.0.

¦ Copyright IBM Corp. 1985, 1991

6.12.9 XESetFlushGC

```
int (*XESetFlushGC(display, extension, proc))()
      Display *display;
      int extension;
      char *(*proc)();
```

display Specifies a display.

extension Specifies an extension number.

proc Specifies a routine to call when an I/O error occurs.

The **XESetFlushGC** defines a procedure to call when a GC in cache needs to be updated in the server.

¦ Copyright IBM Corp. 1985, 1991

6.12.10 XESetFreeFont

```
int (*XESetFreeFont(display, extension, proc))()
      Display *display;
      int extension;
      int (*proc)();
```

display Specifies a display.

extension Specifies an extension number.

proc Specifies a routine to call when a font is freed.

XESetFreeFont defines a procedure to call when **XFreeFont** is called. It returns any previously defined procedure, usually **NULL**.

When **XFreeFont** is called, **XESetFreeFont** is called with these arguments:

```
(*proc)(display, fs, codes)
        Display *display;
        XFontStruct *fs;
        XExtCodes *codes;
```

¦ Copyright IBM Corp. 1985, 1991

6.12.11 XESetFreeGC

```
int (*XESetFreeGC(display, extension, proc)()
      Display *display;
      int extension;
      int (*proc)();
```

display Specifies the display.

extension Specifies the extension number.

proc Specifies the routine to call when a **GC** is freed.

XESetFreeGC defines a procedure to call when a **GC** is freed. It returns any previously defined procedure, usually **NULL**.

When a **GC** is freed, **XESetFreeGC** is called with the following arguments:

```
(*proc)(display, gc, codes)
        Display *display;
        GC gc;
        XExtCodes *codes;
```

¦ Copyright IBM Corp. 1985, 1991

6.12.12 XESetWireToEvent

```
int (*XESetWireToEvent(display, event_number, proc))()
      Display *display;
      int event_number;
      int (*proc)();
```

display Specifies a display.

event_number Specifies a protocol event routine to replace with the conversion routine.

proc Specifies a routine to call when converting the event.

XESetWireToEvent defines a procedure to call when an event is to be converted from wire format (**xEvent** in **<X11/Xproto.h>**) to host format (**XEvent** in **<X11/Xlib.h>**).

XESetWireToEvent returns any previously defined procedure.

Note: The host event structure size cannot be bigger than the size of the **XEvent** union of structures.

When **Xlib** needs to convert an event from wire format to natural host format, **XESetWireToEvent** is called with the following arguments:

```
(*proc)(display, re, event)
        Display *display;
        XEvent *re;
        xEvent event;
```

The argument *re* is a pointer to where the host format event should be stored.

The *event* argument is the 32-byte wire event structure.

In the **XEvent** structure, *type* must be the first field and *window* must be the second field. Copy the *type* field with the type specified for the **xEvent** structure. Copy all other fields from the **xEvent** structure (wire format) to the **XEvent** structure (host format). For further information on **XEvent**, see Chapter 1, "Using X-Windows" in topic 1.0.

¦ Copyright IBM Corp. 1985, 1991

6.12.13 XFreeExtensionList

XFreeExtensionList(*list***)**
char ****list;

list Specifies the allocated memory to be freed.

XFreeExtensionList frees the memory allocated by **XListExtensions**.

¦ Copyright IBM Corp. 1985, 1991

6.12.14 XListExtensions

```
char *XListExtensions(display, nextensions)
      Display *display;
      int *nextensions;
```

display Specifies the display.

nextensions Specifies the extensions supported by the server.

XListExtensions returns a list of all extensions supported by the server.

¦ Copyright IBM Corp. 1985, 1991

6.12.15 XMaxRequestSize

long XMaxRequestSize(*display*)
Display **display*

display Specifies the connection to the X Server.

XMaxRequestSize returns the maximum request size (4-byte units) supported by the server. Single protocol requests to the server cannot be any longer than this. Extensions should be designed so that long protocol requests can be split up into smaller requests. The protocol guarantees the maximum request size to be no smaller than 5096 unit (16384 bytes).

¦ Copyright IBM Corp. 1985, 1991

6.12.16 XQueryExtension

Bool XQueryExtension(*display, name, major_opcode, first_event, first_error***)**
**Display ***display;
**char ***name;
**int ***major_opcode;
**int ***first_event;
**int ***first_error;

display	Specifies the display.
name	Specifies the name of an extension in the form of a case-sensitive ASCII string.
major_opcode	Specifies the returned major opcode for the named extension if it has one or zero.
first_event	Specifies the returned base event type code for the named extension or zero.
first_error	Specifies the returned base error code for the named extension or zero.

XQueryExtension determines if the named extension is present. Any minor opcode, request formats, the format of the events, and the format of additional data in errors are specific to the extension.

¦ Copyright IBM Corp. 1985, 1991

6.13 Using AIX Extensions

AIX X-Windows provides the following extensions:

Polymarkers suppor

Includes **XSetPolyMarker**, **XDrawPolyMarker**, and **XDrawPolyMarkers**.

Note: The following extensions are supported on the RT only.

Lighted Programmable Function Key (Lpfk) suppor

Includes **XActivateAutoLoad**, **XStopAutoLoad**, **XSetLpfkAttributes**, **XGetLpfkAttributes**, **XSetLpfkControl**, **XGetLpfkControl**, **XSelectLpfkInput**, **XSelectLpfk**, **XSelectDeviceInput**, **XSetDeviceInputFocus**, and **XGetDeviceInputFocus**.

Dial (valuator) suppor

Includes **XSetDialAttributes**, **XGetDialAttributes**, **XSetDialControl**, **XGetDialControl**, **XSelectDialInput**, **XSelectDial**, **XSelectDeviceInput**, **XSetDeviceInputFocus**, and **XGetDeviceInputFocus**.

Event handling routines for lpfk and dial event

Includes **XAIXWindowEvent**, **XAIXCheckWindowEvent**, **XAIXMaskEvent**, **XAIXCheckMaskEvent**, **XAIXCheckTypedEvent**, and **XAIXCheckTypedWindowEvent**.

Device Control and Asynchronous input support

Includes **XListInputDevices**, **XEnableInputDevice**, **XDisableInputDevice**, **XAsyncInput**, **XGetAIXInfo**, **XQueryInputDevice**, **XActivateAutoLoad**, **XStopAutoLoad**, **and XQueryAutoLoad**.

¦ Copyright IBM Corp. 1985, 1991

6.14 Lpfk and Dial Extensions

The lpfk and dial (or valuator) devices operate in two modes, **AutoLoad** and **EventReport**. These modes are mutually exclusive. The X Server automatically installs the attributes of dial and lpfk when in **AutoLoad** mode. Under **EventReport** mode, the client is responsible for downloading the attributes of dial and lpfk into the X Server. The server starts up with **EventReport** mode.

Xlib provides routines to change the dial control or get the current dial control parameter. The dial control parameter is dial granularity. Routines for changing the on/off of lpfk keypress input and keylight output are also provided.

The X Server can report **LPFKeyPress** events to a client when a Lighted Programmable Function Key (LPFKey) is pressed. To receive **LPFKeyPress** events in a client application, pass a window ID and **LPFKeyPressMask** as the *event_mask* argument to **XSelectLpfkInput**.

An **LPFKeyPress** event has an event type of **LPFKeyPress** and an associated structure name of **XLPFKeyPressedEvent**.

The X Server can report **DialRotate** events to a client when a dial is rotated. To receive **DialRotate** events in a client application, pass a window ID and **DialRotateMask** as the *event_mask* argument to **XSelectDialInput**.

DialRotate events are generated like **KeyPress** events. They have an event type of **DialRotate** and an associated structure name of **XDialRotatedEvent**.

Subtopics

¦ Copyright IBM Corp. 1985, 1991

6.14.1 Processing Input Extension Event

The event types reported to a client application during event processing depend on the event masks in the *event_mask* argument of **XSelectDeviceInput**. Processing descriptions include explanations of the structure or structures associated with the event. All the event structures contain *type* and *display* fields.

The following table lists the event mask, the associated event type or types, and the structure name associated with the event type.

Event Mask	**Event Type**	**Structure**
LPFKeyPressMask	LPFKeyPress	XLPFKeyPressedEvent
DialRotateMask	DialRotate	XDialRotatedEvent
AIXFocusChangeMask	AIXFocusIn AIXFocusOut	XAIXFocusInEvent XAIXFocusOutEvent
AIXDeviceMapChangeMask	AIXDeviceMappingNotify	XAIXDeviceMappingEv

See "XSelectDeviceInput" in topic 6.15.22 and "Processing Dial and Lpfk Events" in topic 6.14.1.1 for information on the event masks in **XSelectDeviceInput**.

Subtopics
6.14.1.1 Processing Dial and Lpfk Events

Copyright IBM Corp. 1985, 1991

6.14.1.1 Processing Dial and Lpfk Events

The X Server reports **LPFKeyPress** events to clients that need to know when an **LPFKey** is pressed. It also reports **DialRotate** events to clients that need to know when a dial is rotated.

To receive **LPFKeyPress** events in a client application, pass a window ID and **LPFKeyPressMask** as as the *event_mask* to **XSelectLpfkInput**.

To receive **DialRotate** events in a client application, pass a window ID and **DialRotateMask** as the *event_mask* to **XSelectDialInput**.

The source of the event is the smallest window containing the pointer. The window used by the X Server to report these events depends on its position in the window hierarchy and whether any intervening window prohibits the generation of these events.

The X Server searches the window hierarchy, starting with the source window until it locates the first window specified by a client as having an interest in these events. If one of the intervening windows has its *do_not_propagate_mask* set to prohibit generation of the event type, the event of those types are suppressed. Clients can modify the window used for reporting with **XSetDeviceInputFocus** routine.

The structures associated with these events are **XLFPKeyPressedEvent** and **XDialRotatedEvent**. These structures have the following fields:

The *window* which is the ID of the window on which the event was generated. It is referred to as the *event window*. The X Server uses this window to report the event.

The *root* which is the window ID of the source window or the root window.

The *x_root* and *y_root* coordinates, which are relative to the origin of the root window origin at the time of the event, and which are set to the pointer.

The *same_screen* which indicates whether the event window is on the same screen as the root window. This can be:

- **True**, which indicates that the event window and the root window are on the same screen.

- **False**, which indicates that the event window and the root window are not on the same screen.

The *subwindow* which is set to the child of the event window that is an ancestor of or is the source field if the source window is an inferior of the event window. Otherwise, the X Server sets *subwindow* to **None**.

The *time* which is set to the time (in milliseconds) when the event was generated since the server reset.

The *x* and *y* coordinates, which are relative relative to the origin of the event window if the event window is on the same screen as the root window. Otherwise, these coordinates are zero.

The *state* which indicates the state of the pointer buttons and modifier keys just prior to the event. The pointer buttons can be the

¦ Copyright IBM Corp. 1985, 1991

bitwise inclusive OR of one or more of the following button or modifier key masks:

Button1Mask	**Mod1Mask**	**ShiftMask**
Button2Mask	**Mod2Mask**	**LockMask**
Button3Mask	**Mod3Mask**	**ControlMask**
Button4Mask	**Mod4Mask**	
Button5Mask	**Mod5Mask**.	

Each structure also has the following unique components.

The **XLPFKeyPressedEvent** structure contains a *keycode* which is set to a number that represents a physical key on the lpfk that ranges from 0 to 31.

The **XDialRotatedEvent** structure contains a *dialnum*, which represents the dial that has been rotated, and *dialval*, which represents the difference between the current dial value and the last dial value. For clockwise rotation, this value is positive. For counterclockwise rotation, this value is negative.

¦ Copyright IBM Corp. 1985, 1991

6.14.2 Processing Dial and Lpfk Input Focus Events

This section describes the processing that occurs for the input focus events **AIXFocusIn** and **AIXFocusOut**. The X Server reports **AIXFocusIn** or **AIXFocusOut** events to clients requiring information about when the dial or lpfk input focus changes. The dial or lpfk is always attached to a window, which is usually the root window or a top-level window called the *focus window*. The focus window and the position of the pointer determines the window that receives dial or lpfk input.

To receive **AIXFocusIn** and **AIXFocusOut** events in a client application, pass a window ID and **AIXFocusChangeMask** as the *ext_event_mask* argument to **XSelectDeviceInput**.

The fields of the **XAIXFocusInEvent** and **XAIXFocusOutEvent** structures associated with these events include the following:

The *window* which is the ID of the window on which the **AIXFocusIn** or **AIXFocusOut** event was generated. The X Server uses this window to report the event.

The *mode* which is **NotifyNormal**.

The *devtype* which is the focus device.

The *detail* which indicates the notify detail. It can be one of the following:

NotifyAncestor	**NotifyVirtual**
NotifyInferior	**NotifyNonlinear**
NotifyNonlinearVirtual	**NotifyPointer**
NotifyPointerRoot	**NotifyDetailNone**

Subtopics
6.14.2.1 Processing AIXFocus Events
6.14.2.2 Processing AIXDeviceMappingNotify Events

¦ Copyright IBM Corp. 1985, 1991

6.14.2.1 Processing AIXFocus Events

Focus events are identified by **XAIXFocusInEvent** or **XAIXFocusOutEvent** structures whose *mode* field is **NotifyNormal**. The X Server processes normal focus events according to the following scenarios:

When the focus moves from window A to window B and A is an inferior of B with the pointer in window P, the X Server generates:

An **AIXFocusOut** event on window *A* that has the **XAIXFocusOutEvent** structure with **NotifyAncestor** as the *detail* field.

An **AIXFocusOut** event on each window between window A and window B exclusive that has the **XAIXFocusOutEvent** structure with **NotifyVirtual** as the *detail* field.

An **AIXFocusIn** event on window B that has the **XAIXFocusOutEvent** structure with **NotifyInferior** as the *detail* field.

An **AIXFocusIn** event on each window below window B down to and including window P if window P is an inferior of window B, but is not window A, and is not an inferior of window A that has the **XAIXFocusInEvent** structure with **NotifyInferior** as the the *detail* field.

When the focus moves from window A to window B and B is an inferior of A with the pointer in window P, the X Server generates:

An **AIXFocusOut** event on each window from window P up to, but not including window A (in that order), if window P is an inferior of window A, but is not window A, and window P is not an inferior of window B nor an ancestor of window B that has the **XAIXFocusOutEvent** structure with **NotifyPointer** as the *detail* field.

An **AIXFocusOut** event on window A that has the **XAIXFocusOutEvent** structure with **NotifyInferior** as the *detail* field.

An **AIXFocusIn** event on each window between window A and window B exclusive, that has the **XAIXFocusInEvent** structure with **NotifyVirtual** as the *detail* field.

An **AIXFocusIn** event on window B that has the **XAIXFocusInEvent** structure with **NotifyAncestor** as the *detail* field.

When the focus moves from window A to window B and window C is their least common ancestor, and with the pointer in window P, the X Server generates:

An **AIXFocusOut** event on each window from window P up to but not including window A and window P is an inferior of window A that has the **XAIXFocusOutEvent** structure with **NotifyPointer** as the *detail* field.

An **AIXFocusOut** event on window A that has the **XAIXFocusOutEvent** structure with **NotifyNonlinear** as the *detail* field.

An **AIXFocusOut** event on each window between window A and window C exclusive, that has the **XAIXFocusOutEvent** structure with **NotifyNonlinearVirtual** as the *detail* field.

An **AIXFocusIn** event on each window between C and B exclusive that has

¦ Copyright IBM Corp. 1985, 1991

the **XAIXFocusInEvent** structure with **NotifyNonlinearVirtual** as the *detail* field.

An **AIXFocusIn** event on window B that has the **XAIXFocusInEvent** structure with **NotifyNonlinear** as the *detail* field.

An **AIXFocusIn** event on each window below window B down to and including window P and window P is an inferior of window B that has the **XAIXFocusInEvent** structure with **NotifyPointer** as the *detail* field.

If the focus window is **PointerRoot** (events sent to the window under the pointer) or **None** in **XSetDeviceInputFocus**, when the focus moves from window A to **PointerRoot** or **None** with the pointer in window P, the X Server generates:

An **AIXFocusOut** event on each window from window P up to, but not including window A, and window P is an inferior of window A, that has the **XAIXFocusOutEvent** structure with **NotifyPointer** as the *detail* field.

An **AIXFocusOut** event on window A that has the **XAIXFocusOutEvent** structure with **NotifyNonlinear** as the *detail* field.

An **AIXFocusOut** event on each window above window A up to, and including its root, and window A is not a root window, that has the **XAIXFocusOutEvent** structure with **NotifyNonlinearVirtual** as the *detail* field.

An **AIXFocusIn** event on the root window of all screens that has the **XAIXFocusInEvent** structure with **NotifyPointerRoot** or **NotifyDetailNone** as the *detail* field.

An **AIXFocusIn** event on each window from the root of window P down to and including window P, (new focus **PointerRoot**) that has **XAIXFocusInEvent** structure with **NotifyPointerRoot** as the *detail* field.

When the focus moves from **PointerRoot** (events sent to the window under the pointer) or **None** to window A, with the pointer in window P, the X Server generates:

An **AIXFocusOut** event on each window from window P up to, and including the root of window P, (the old focus **PointerRoot**) that has the **XAIXFocusOutEvent** structure with **NotifyPointerRoot** as the *detail* field.

An **AIXFocusOut** event on all root windows that has the **XAIXFocusEvent** structure with **NotifyPointerRoot** or **NotifyDetailNone** as the *detail* field.

An **AIXFocusIn** event on each window from the root of window A down to, but not including window A, and window A is not a root window, that has the **XAIXFocusInEvent** structure with **NotifyNonlinearVirtual** as the *detail* field.

An **AIXFocusIn** event on window A that has the **XAIXFocusInEvent** structure with **NotifyNonlinear** as the *detail* field.

An **AIXFocusIn** event on each window below window A down to and including window P, and window P is an inferior of window A that has the **XAIXFocusInEvent** structure with the **NotifyPointer** as the *detail*

¦ Copyright IBM Corp. 1985, 1991

field.

When the focus moves from **PointerRoot** (events sent to the window under the pointer) to **None** (or vice versa), with the pointer in window *P*, the X Server generates:

An **AIXFocusOut** event on each window from window P, up to and including the root of window P, (old focus is **PointerRoot**) that has the **XAIXFocusOutEvent** structure with **NotifyPointerRoot** as the *detail* field.

An **AIXFocusOut** event on all root windows that has the **XAIXFocusOutEvent** structure with **NotifyPointerRoot** or **NotifyDetailNone** as the *detail* field.

An **AIXFocusIn** event on all root windows that has the **XAIXFocusInEvent** structure with **NotifyDetailNone** or **NotifyPointerRoot** as the *detail* field.

An **AIXFocusIn** event on each window from the root of window P down to and including window P (new focus is **PointerRoot**) that has the **XAIXFocusInEvent** structure with **NotifyPointerRoot** as the *detail* field.

¦ Copyright IBM Corp. 1985, 1991

6.14.2.2 Processing AIXDeviceMappingNotify Events

The X Server reports **AIXDeviceMappingNotify** events to all clients. This event type is generated when a client application uses:

XSetDialControl which indicates that new dial granularity has been set.

XSetLpfkControl which indicates that new lpfk input/output settings.

The fields of the **XAIXMappingEvent** structure associated with **XAIXDeviceMappingNotify** event include *request, dialmask, lpfkmask* and *lightmask*.

The *request* indicates the kind of mapping change that occurred. It can be **AIXMappingDial** or **AIXMappingLpfk**.

If *request* is **AIXMappingDial**, the dial granularity is changed. If *request* is **AIXMappingLpfk**, the lpfk input/output setting is changed.

The *dialmask, lpfkmask* and *lightmask* indicate the setting value.

¦ Copyright IBM Corp. 1985, 1991

6.15 AIX Extensions

The AIX protocol extensions are described in the this portion of this chapter. These extensions are in alphabetical order.

Subtopics

¦ Copyright IBM Corp. 1985, 1991

6.15.1 XActivateAutoLoad

XActivateAutoLoad(*display***)**
Display **display*;*

display Specifies the connection to the X Server.

XActivateAutoLoad activates loading dial granularity and lpfk light setting. When the pointer device crosses the boundary of a window with dial or lpfk control settings, the X Server loads the dial or lpfk control setting automatically. It does not generate events to the client. If **AutoLoad** mode is on, focus is not supported on either device and it behaves like a cursor changing shape when it crosses a window boundary.

This is supported on the RT only.

¦ Copyright IBM Corp. 1985, 1991

6.15.2 XAIXCheckMaskEvent

Bool XAIXCheckMaskEvent(*display, ext_event_mask, event_return***)**
Display ****display*;**
unsigned long *ext_event_mask*;
XEvent **event_return*;

display Specifies the connection to the X Server.

ext_event_mask Specifies the extension event mask, which is the bitwise inclusive OR of one or more of the valid extension event mask bits.

event_return Specifies a client-supplied structure that receives a copy of the associated structure event that matches the extension event.

XAIXCheckMaskEvent does the following:

It removes the next event, if any, that matches the extension even mask.

It does not block

It returns a Boolean indicating if the event was found

XAIXCheckMaskEvent searches the event queue, then the events available for the first event that matches the specified mask and is available on the server connection.

When **XAIXCheckMaskEvent** finds a match, it removes the event, copies it into the specified **XEvent** structure, and returns **True**. Other events stored in the queue are not discarded. If the event is not in the queue, **XAIXCheckMaskEvent** flushes the output buffer and returns **False**.

This is supported on the RT only.

¦ Copyright IBM Corp. 1985, 1991

6.15.3 XAIXCheckTypedEvent

```
int XAIXCheckTypedEvent(display, event_type, event_return)
      Display *display;
      int event_type;
      XEvent *event_return;
```

display Specifies the connection to the X Server.

event_type Specifies the extension event type to be compared.

event_return Specifies a client-supplied structure that receives a copy of the associated structure event that matches the extension event.

XAIXCheckTypedEvent returns the next event in the queue that matches an event type. It searches the event queue, then the events available, for an event that matches the specified type.

When **XAIXCheckTypedEvent** finds a match, it returns the event structure to the specified **XEvent** structure and returns **True**. Other events in the queue are not discarded. If the event is not available, **XAIXCheckTypedEvent** flushes the output buffer and returns **False**.

This is supported on the RT only.

¦ Copyright IBM Corp. 1985, 1991

6.15.4 XAIXCheckTypedWindowEvent

int XAIXCheckTypedWindowEvent(*display, window, event_type, event_return***)**
Display ****display*;**
Window *window***;**
int *event_type***;**
XEvent **event_return*;

display	Specifies the connection to the X Server.
window	Specifies the window ID.
event_type	Specifies the event type to compare.
event_return	Specifies a client-supplied structure that receives a copy of the associated structure event that matches the extension event.

XAIXCheckTypedWindowEvent returns the next matched event in the queue for the specified window. First, it searches the event queue, then any events available on the server for an event that matches the specified type and window.

When **XAIXCheckTypedWindowEvent** finds a match, it removes the event from the queue, copies it into the specified **XEvent** structure and returns **True**. Other events in the queue are not discarded. If the event is not available, **XAIXCheckTypedWindowEvent** flushes the output buffer and returns **False**.

This is supported on the RT only.

¦ Copyright IBM Corp. 1985, 1991

6.15.5 XAIXCheckWindowEvent

Bool XAIXCheckWindowEvent(*display, window, ext_event_mask, event_return***)**
**Display ****display*;
Window *window*;
unsigned long *ext_event_mask*;
XEvent **event_return*;

display Specifies the connection to the X Server.

window Specifies the window ID for the window with the next matched extension event to be removed.

ext_event_mask Specifies the event mask which is the bitwise-inclusive OR of one or more of the valid extension event mask bits.

event_return Specifies a client-supplied structure that receives a copy of the associated structure event that matches the extension event.

XAIXCheckWindowEvent removes the next extension event that matches both the passed window and the passed mask. This routine does not block. It returns a zero or a one to indicate if the event was returned.

XAIXCheckWindowEvent searches the event queue first, then, it searches the events available on the server connection for the first event that matches the specified window and extension event mask. When a match is found, **XAIXCheckWindowEvent** removes that extension event, copies it into the specified **XEvent**, and returns **True**. The other events stored in the queue are not discarded. If an event is not found, **XAIXCheckWindowEvent** flushes the output buffer and returns **False**.

This is supported on the RT only.

¦ Copyright IBM Corp. 1985, 1991

6.15.6 XAIXMaskEvent

XAIXMaskEvent(*display, ext_event_mask, event_return***)**
Display ****display*;**
unsigned long *ext_event_mask*;
XEvent **event_return*;

display Specifies the connection to the X Server.

ext_event_mask Specifies the extension event mask which is the bitwise inclusive OR of one or more of the valid event mask bits.

event_return Specifies a client-supplied structure that receives a copy of the associated structure event that matches the extension event.

XAIXMaskEvent removes the next event that matches an extension event mask. It searches the event queue for the extension events associated with the specified mask. When **XAIXMaskEvent** finds a match, it removes that event and copies it into the specified **XEvent** structure. Other events stored in the queue are not discarded. If the extension event requested is not in the queue, **XAIXWindowEvent** flushes the output buffer and blocks until one is received.

Note: **XAIXWindowEvent** remains blocked if extension events do not come across the wire.

This is supported on the RT only.

¦ Copyright IBM Corp. 1985, 1991

6.15.7 XAIXWindowEvent

XAIXWindowEvent(*display, window, ext_event_mask, event_return***)**
Display ***display;**
Window *window*;**
unsigned long *ext_event_mask*;**
XEvent **event_return*;

display	Specifies the connection to the X Server.
window	Specifies the window ID with the next matched extension event to be removed.
ext_event_mask	Specifies the event mask which is the bitwise inclusive OR of one or more of the valid extension event mask bits.
event_return	Specifies a client-supplied structure that receives a copy of the associated structure event that matches the extension event.

XAIXWindowEvent removes the next event that matches both a window and an extension event mask. It searches the event queue for an event that matches both the specified window and the extension event mask. When it finds a match, it removes that extension event from the queue and copies it into the specified **XEvent** structure. Other events stored in the queue are not discarded. If the extension event requested is not in the queue, **XAIXWindowEvent** flushes the output buffer and blocks until one is received.

Note: **XAIXWindowEvent** remains blocked if extension events do not come across the wire.

This is supported on the RT only.

¦ Copyright IBM Corp. 1985, 1991

6.15.8 XAsyncInput

int
(*XAsyncInput(*display, InputHandler***))()**
Display ***display**;
int *(*InputHandler)*();

display Specifies the connection to the X Server.

InputHandler Specifies the procedure called for every event reported by the X Server.

XAsyncInput sets up asynchronous input support. Once it is set up, **XAsyncInput()** is called for each event received by the client. If the input handler returns zero, the event is discarded. If the input handler returns one, the event is queued and another X-Windows calls can be used to remove it from the queue.

InputHandler(display, event, level)
Display ***display**;
XEvent ***event**;
int *level*;

display Specifies the connection to the X Server.

event Specifies the event from the X Server.

level Specifies the level of asynchronous input.

The input handler can make X-Windows calls with the *level* parameter indicating the depth of nesting. If an application is on an **XNextEvent()**, the input handler still sees every event first.

This is supported on the RT only.

¦ Copyright IBM Corp. 1985, 1991

6.15.9 XDisableInputDevice

XDisableInputDevice(*display, device***)**
Display ***display;
int *device;*

display Specifies the connection to the X Server.

device Specifies the input device.

XDisableInputDevice disables the specified input device so that the X Server cannot report events to it.

XDisableInputDevice can generate the error **AIXBadDevice**.

This is supported on the RT only.

¦ Copyright IBM Corp. 1985, 1991

6.15.10 XDrawPolyMarker

XDrawPolyMarker(*display, window, gc, x, y***)**
Display ***display**;
Window *window*;
GC *gc*;
int *x, y*;

display Specifies the connection to the X Server.

window Specifies the window ID.

gc Specifies the graphics context.

x,y Specifies the *x* and *y* coordinates where the marker will be drawn.

XDrawPolyMarker draws a single marker into the specified window with extended graphics context. It is not affected by the tile or stipple in the graphics context.

XDrawPolyMarker uses the graphics context components *function, plane_mask, foreground, clip_x_origin*, and *clip_y_origin*.

XDrawPolyMarker can generate the errors **BadDrawable**, **BadGC**, and **BadMatch**.

Note: On the PS/2, this routine is in **/usr/lib/liboldX.a**. Use **-loldX** on the command line to link to this library.

This is supported on the RT and the PS/2.

¦ Copyright IBM Corp. 1985, 1991

6.15.11 XDrawPolyMarkers

XDrawPolyMarkers(*display, window, gc, points, npoints, mode***)**
Display ***display**;
Window *window*;
GC *gc*;
XPoint ***points**;
int *npoints*;
int *mode*;

display	Specifies the connection to the X Server.
window	Specifies the window ID.
gc	Specifies the graphics context.
points	Specifies a pointer to an array of points.
npoints	Specifies the number of points in the array.
mode	Specifies the coordinate mode.

XDrawPolyMarkers draws multiple markers in the specified window. It uses the extended graphics context component *marker*.

XDrawPolyMarkers draws multiple markers into the specified window, but is not affected by the tile or stipple in the graphics context. Specify an array of **XPoint** structures. Each array contains *x* and *y* coordinates. **XDrawPolyMarkers** draws the markers in the order listed in the array.

The *mode* argument indicates whether the points are relative to the window origin or to the previous point. If *mode* is:

CoordModeOrigin, all points after the first are relative to the window origin. The first point is always relative to the window origin.

CoordModePrevious, all points after the first are relative to the previous point.

XDrawPolyMarkers uses the graphics context components *function, plane_mask, foreground, clip_x_origin*, and *clip_y_origin*.

XDrawPolyMarkers can generate the errors **BadDrawable**, **BadGC**, **BadValue**, and **BadMatch**.

Note: On the PS/2, this routine is in **/usr/lib/liboldX.a**. Use **-loldX** on the command line to link to this library.

This is supported on the RT and the PS/2.

¦ Copyright IBM Corp. 1985, 1991

6.15.12 XEnableInputDevice

```
int XEnableInputDevice(display, device)
      Display *display;
      int     device;
```

display Specifies the connection to the X Server.

device Specifies the input device.

XEnableInputDevice enables event input by allowing the X Server to report events from the specified input device.

XEnableInputDevice can generate the error **AIXBadDevice**.

This is supported on the RT only.

¦ Copyright IBM Corp. 1985, 1991

6.15.13 XGetAIXInfo

XGetAIXInfo(*display, keyboardid, vrmid, displayid***)**
Display ***display;**
int ***keyboardid,**
int ***vrmid;**
int **displayid**;

display Specifies the connection to the X Server.

keyboardid Returns current keyboard ID.

vrmid Returns the current display VRMID.

displayid Returns display model number.

XGetAIXInfo gets the current AIX specific information.

XGetAIXInfo returns the current value of *keyboardid, vrmid* and *displayid*. Check **<X11/AIXlib.h>** for the meaning of returned symbols definition.

This is supported on the RT only.

¦ Copyright IBM Corp. 1985, 1991

6.15.14 XGetDeviceInputFocus

XGetDeviceInputFocus(*display, device, focus_return, revert_to_return***)**
**Display ***display*;
int *device*;
**Window ***focus_return*;
int **revert_to_return*;

display Specifies the connection to the X Server.

device Specifies the input device. **DEVDIAL** and **DEVLPFK** are supported.

focus_return Returns the focus window ID, **PointerRoot** or **None**.

revert_to_return Returns the current focus state as **RevertToParent**, **RevertToPointerRoot**, or **RevertToNone**.

XGetDeviceInputFocus returns the current dial or lpfk input focus state and focus window ID.

XGetDeviceInputFocus can generate the error **AIXBadDevice**.

This is supported on the RT only.

¦ Copyright IBM Corp. 1985, 1991

6.15.15 XGetDialAttributes

XGetDialAttributes(*display, window, dialmask, resolution, num***)**
```
      Display  *display;
      Window  window;
      unsigned long dialmask;
      char *resolution;
      int num;
```

display	Specifies the connection to the X Server.
window	Specifies the window ID.
dialmask	Specifies the dial mask which is the bitwise inclusive OR of one or more valid dial mask bits.
resolution	Specifies an array of dial resolutions.
num	Specifies the number of the 1-bit in *dialmask*.

The **XGetDialAttributes** returns the dial resolutions specified on *dialmask* of the specified window. Allocate the resolution array, 1-byte for each dial.

XGetDialAttributes can generate the errors **BadValue** and **BadWindow**.

This is supported on the RT only.

¦ Copyright IBM Corp. 1985, 1991

6.15.16 XGetDialControl

XGetDialControl(*display, dialmask, resolution, num***)**
**Display ***display;
int *dialmask;*
**char ***resolution;*
int *num;*

display Specifies the connection to the X Server.

dialmask Specifies the dial mask bits.

resolution Returns dials resolution.

num Specifies the number of the 1-bit on *dialmask*.

XGetDialControl gets the current dial granularity. It returns the dial resolutions specified on *dialmask*. Allocate the resolution array, 1-byte for each dial.

This is supported on the RT only.

¦ Copyright IBM Corp. 1985, 1991

6.15.17 XGetLpfkAttributes

XGetLpfkAttributes(*display, window, lpfkmask, lightmask***)**
Display ***display**;
Window *window*;
unsigned long ***lpfkmask**;
unsigned long **lightmask*;

display Specifies the connection to the X Server.

window Specifies the window ID.

lpfkmask Specifies the **LPFKey** mask which is the bitwise inclusive OR of one or more valid **LPFKey** mask bits.

lightmask Specifies the **LPFKey** mask, which is the bitwise inclusive OR of one or more valid **LPFKey** mask bits.

XGetLpfkAttributes gets the **LPFKey** input and output mask.

Valid **LPFKey** mask bits are **LPFKeyMask0**, **LPFKeyMask2** through **LPFKeyMask31**.

XGetLpfkAttributes can generate the errors **BadValue** and **BadWindow**.

This is supported on the RT only.

¦ Copyright IBM Corp. 1985, 1991

6.15.18 XGetLpfkControl

XGetLpfkControl(*display, lpfkmask, lightmask***)**
**Display ***display;
**unsigned long ***lpfkmask;
unsigned long **lightmask;*

display Specifies the connection to the X Server.

lpfkmask Returns the current lpfk input mask setting.

lightmask Returns the current lpfk output mask setting.

XGetLpfkControl returns the current input and output setting of lpfk.

This is supported on the RT only.

¦ Copyright IBM Corp. 1985, 1991

6.15.19 XListInputDevices

AIXInputDeviceInfo *XListInputDevices(*display, ndevices, enabled***)**
**Display ***display;
**int ***ndevices;
Bool **enabled;*

display Specifies the connection to the X Server.

ndevices Returns the number of devices currently supported by the X Server.

enabled Returns the number of currently enabled devices.

XListInputDevices obtains a list of devices currently supported by the X Server. It finds out which devices are available and returns a pointer to a list of device structures that were allocated by the routine.

Use **Xfree** to free the memory after this function has completed.

The **AIXInputDeviceInfo** data structure is:

```
typedef struct {
    short deviceid;
    short state;
    int size;
    char *devinfo;
} AIXInputDeviceInfo;
```

deviceid Specifies device ID which are defined in **<X11/XAIX.h>**.

state Specifies the on or off state of the specified device.

size Specifies the size of *devinfo*.

devinfo Specifies a pointer to a device private structure. It always returns **NULL**.

This is supported on the RT only.

¦ Copyright IBM Corp. 1985, 1991

6.15.20 XQueryAutoLoad

XQueryAutoLoad (*display, onoff*)
Display ***display**;
int ***onoff**;

display Specifies the connection to the X Server.

onoff Returns the event mode of **dial** and **lpfk**.

XQueryAutoLoad returns the current event mode of dial and lpfk. If *onoff* is:

One, **Autoload** mode is on.

Zero, **EventReport** mode is on.

This is supported on the RT only.

¦ Copyright IBM Corp. 1985, 1991

6.15.21 XQueryInputDevice

XQueryInputDevice (*display, device, onoff*)
Display **display*;
int *device*;
int **onoff*;

display Specifies the connection to the X Server.

device Specifies the device ID.

onoff Returns the on or off status of the specified device.

XQueryInputDevice returns the current status of the specified device. When *onoff* is:

One, the device is on

Zero, the device is off

This is supported on the RT only.

¦ Copyright IBM Corp. 1985, 1991

6.15.22 XSelectDeviceInput

XSelectDeviceInput(*display, device, window, ext_event_mask***)**
Display ***display;**
int *device*;
Window *window*;
unsigned long *ext_event_mask*;

display Specifies the connection to the X Server.

device Specifies the device ID.

window Specifies the window ID. Client applications interested in an extension event for a particular window pass that window ID.

ext_event_mask Specifies the extension event mask, which is the bitwise inclusive OR of one or more of the valid extension event mask bits.

XSelectDeviceInput requests the X Server to report the events associated with the event masks from the specified device.

With this routine, events are reported relative to a window. If a window is not interested in an event, the events are reported to the closest ancestor that is interested.

A call to **XSelectDeviceInput** overrides any previous call to **XSelectDeviceInput** for the same window from the same client, but not for other clients. Different clients can select events on the same window. Events are reported to all interested clients. The extension event mask is a shared resource. Do not override the extension mask.

XSelectDeviceInput can generate the errors **BadValue** and **BadWindow**.

This is supported on the RT only.

¦ Copyright IBM Corp. 1985, 1991

6.15.23 XSelectDial

XSelectDial(*display, window, dialmask***)**
Display ***display;**
Window *window*;
unsigned long *dialmask*;

display Specifies the connection to the X Server.

window Specifies the window ID. Client applications interested in an extension event for a particular window pass that window ID.

dialmask Specifies the dial mask, which is the bitwise inclusive OR of one or more valid dial mask bits.

XSelectDial associates a dial with a window ID. Usually, when **XSelectDialInput** is executed, it is followed by **XSelectDial**.

XSelectDial can generate the errors **BadValue** and **BadWindow**.

This is supported on the RT only.

¦ Copyright IBM Corp. 1985, 1991

6.15.24 XSelectDialInput

XSelectDialInput(*display, window, ext_event_mask***)**
**Display ***display;
Window *window;*
unsigned long *ext_event_mask;*

display Specifies the connection to the X Server.

window Specifies the window ID. Client applications interested in an extension event for a particular window pass that window ID.

ext_event_mask Specifies the extension event mask, which is the bitwise inclusive OR of one or more of the valid extension event mask bits.

XSelectDialInput requests the server to report events associated with the events masks passed to the *ext_event_mask* argument.

With this routine, events are reported relative to a window. If a window is not interested in an event, the events are reported to the closest ancestor that is interested.

A call to **XSelectDialInput** overrides any previous call to **XSelectDialInput** for the same window from the same client, but not for other clients. Different clients can select events on the same window. Events are reported to all interested clients.

XSelectDialInput can generate the errors **BadValue** and **BadWindow**.

This is supported on the RT only.

¦ Copyright IBM Corp. 1985, 1991

6.15.25 XSelectLpfk

XSelectLpfk(*display, window, lpfkmask, lightmask***)**
**Display ***display*;
Window *window*;
unsigned long *lpfkmask*;
unsigned long *lightmask*;

display Specifies the connection to the X Server.

window Specifies the window ID. Client applications interested in an extension event for a particular window pass that window ID.

lpfkmask Specifies the **LPFKey** input mask, which is the bitwise inclusive OR of one or more of the valid extension event mask bits.

lightmask Specifies the **LPFKey** light output mask, which is the bitwise inclusive OR of one or more of the valid extension event mask bits.

XSelectLpfk associates **LPFKey** input and output with a window. When **XSelectLpfkInput** is executed, it is followed by **XSelectLpfk** usually.

XSelectLpfk can generate the errors **BadValue** and **BadWindow**.

This is supported on the RT only.

¦ Copyright IBM Corp. 1985, 1991

6.15.26 XSelectLpfkInput

XSelectLpfkInput(*display, window, ext_event_mask***)**
**Display ***display*;
Window *window*;
unsigned long *ext_event_mask*;

display Specifies the connection to the X Server.

window Specifies the window ID. Client applications interested in an extension event for a particular window pass that window ID.

ext_event_mask Specifies the extension event mask which is the bitwise inclusive OR of one or more of the valid extension event mask bits.

XSelectLpfkInput requests the server to report events associated with the events masks passed to the *ext_event_mask* argument.

With this routine, events are reported relative to a window. If a window is not interested in an event, the events are reported to the closest ancestor that is interested.

A call to **XSelectLpfkInput** overrides any previous call to **XSelectLpfkInput** for the same window from the same client, but not for other clients. Different clients can select events on the same window. Events are reported to all interested clients.

XSelectLpfkInput can generate the errors **BadValue** and **BadWindow**.

This is supported on the RT only.

¦ Copyright IBM Corp. 1985, 1991

6.15.27 XSetDeviceInputFocus

XSetDeviceInputFocus(*display, device, focus, revert_to, time***)**
Display ***display**;
int *device*;
Window *focus*;
int *revert_to*;
Time *time*;

display Specifies the connection to the X Server.

device Specifies the input device (**DEVDIAL** or **DEVLPFK**).

focus Specifies the window ID, **PointerRoot** or **None** for setting the input focus.

revert_to Specifies the window the input focus reverts to if the window becomes not viewable. It can be set to **RevertToParent**, **RevertToPointerRoot**, or **RevertToNone**.

time Specifies the time in a timestamp (in milliseconds) or **CurrentTime**.

XSetDeviceInputFocus sets the device input focus and the last-focus-change time. The X Server generates **AIXFocusIn** and **AIXFocusOut** events.

This routine has no effect if the specified time is earlier than the current last-focus-change time or is later than the current X Server time. Otherwise, the last-focus-change time is set to the specified time and **CurrentTime** is replaced by the current X Server time.

If *focus* is:

A window ID, this window becomes the focus window of dial or lpfk. I a generated dial or lpfk event is normally reported to this window or one of its inferiors, the event is still reported normally. Otherwise, the event is reported relative to the focus window.

PointerRoot, the focus window is the root window of whatever screen contains the pointer during each dial or lpfk event. In this case, the *revert_to* argument is ignored.

None, all dial and lpfk events are discarded until a new focus window is set. In this case, the *revert_to* argument is ignored.

The specified focus window must be viewable at the time **XSetDeviceInputFocus** is called. Otherwise, a **BadMatch** error is generated.

If the focus window becomes unviewable later, the X Server evaluates the *revert_to* argument to determine the new focus window. If *revert_to* is:

RevertToParent, the focus reverts to the parent (or the closest viewable ancestor), and the new *revert_to* value is **RevertToNone**.

RevertToPointerRoot or **RevertToNone**, the focus reverts to that value. When the focus reverts, the X Server generates **AIXFocusIn** and **AIXFocusOut** events, but the last-focus-change time is not affected.

XSetDeviceInputFocus can generate the errors **BadMatch**, **BadWindow**, **BadValue**, and **AIXBadDevice**.

¦ Copyright IBM Corp. 1985, 1991

This is supported on the RT only.

¦ Copyright IBM Corp. 1985, 1991

6.15.28 XSetDialAttributes

XSetDialAttributes(*display, window, dialmask, resolution, num***)**
Display ***display**;
Window *window*;
unsigned long *dialmask*;
char ***resolution**;
int *num*;

display Specifies the connection to the X Server.

window Specifies the window ID.

dialmask Specifies the dial mask which is the bitwise inclusive OR of one or more valid dial mask bits.

resolution Specifies an array of dial resolutions.

num Specifies the number of the 1-bit in *dialmask*.

XSetDialAttributes sets dial resolution if **Autoload** mode is set. The resolution array must be as large as the largest numbered dial in the dialmask. The value of a resolution array entry represents the amount of resolution that:

Has a value of 2 through	
Indicates a resolution of	4, 8, 16, 64, or 128 points per revolution

The valid dial mask bits are **DialMask0** through **DialMask31**.

XSetDialAttributes can generate the errors **BadValue** and **BadWindow**.

This is supported on the RT only.

¦ Copyright IBM Corp. 1985, 1991

6.15.29 XSetDialControl

XSetDialControl(*display, dialmask, resolution, num***)**
**Display ***display;
Bool *dialmask;*
**char ***resolution;*
int *num;*

display Specifies the connection to the X Server.

dialmask Specifies dial mask bits.

resolution Specifies the dial resolution from 2 to 7.

num Specifies the number of the 1-bit in *dialmask*.

XSetDialControl controls the global granularity of a dial input device. It sets the dial granularity if **AutoLoad** is off. The X Server generates **AIXDeviceMappingNotify** events to all clients.

XSetDialControl can generate the errors **BadValue** and **AIXBadMode**.

This is supported on the RT only.

¦ Copyright IBM Corp. 1985, 1991

6.15.30 XSetLpfkAttributes

XSetLpfkAttributes(*display, window, lpfkmask, lightmask***)**
```
      Display  *display;
      Window  window;
      unsigned long lpfkmask;
      unsigned long lightmask;
```

display Specifies the connection to the X Server.

window Specifies the window ID.

lpfkmask Specifies the **Lpfkey** input mask which is the bitwise inclusive OR of one or more valid **Lpfkey** mask bits. This indicates if the X Server reports the lpfk keypress event to a client.

lightmask Specifies the **Lpfkey** output mask which is the bitwise inclusive OR of one or more valid **LPFKey** mask bits.

XSetLpfkAttributes sets **LPFKey** input and output masks. **Autoload** mode must be on.

When the pointer enters a visible window, the **LPFKey** lights are set to reflect those defined for the window. If an **LPFKey** definition does not exist, turn off the lights.

The valid **LPFKey** mask bits are **LPFKeyMask0**, **LPFKeyMask2**, through **LPFKeyMask31**.

XSetLpfkAttributes can generate the errors **BadValue** and **BadWindow**.

This is supported on the RT only.

¦ Copyright IBM Corp. 1985, 1991

6.15.31 XSetLpfkControl

XSetLpfkControl(*display, lpfkmask, lightmask***)**
Display ***display;**
unsigned long *lpfkmask*;
unsigned long *lightmask*;

display Specifies the connection to the X Server.

lpfkmask Specifies **lpfk** key input mask.

lightmask Specifies **lpfk** key output mask.

XSetLpfkControl changes the input and output of **LPFKeys**. It controls the lpfk mechanism of *keylight* and *keypress*.

The *lpfkmask* argument specifies which lpfk *keypress* should be turned on. To receive input for each 1-bit in the lightmask, turn *keypress* on. If 0-bit is specified, the X Server does not generate a **LPFKeyPress** event.

The *lightmask* controls which lpfk keylight should be on. Turn on keylight for each 1-bit in the *lightmask*.

XSetLpfkControl can generate the errors **BadValue** and **AIXBadMode**.

This is supported on the RT only.

¦ Copyright IBM Corp. 1985, 1991

6.15.32 XSetPolyMarker

XSetPolyMarker(*display, gc, marker, x, y***)**
Display ***display**;
GC *gc*;
Pixmap *marker*;
int *x, y*;

display Specifies the connection to the X Server.

gc Specifies the graphics context.

marker Specifies the marker to set for the specified graphics context. Note that the depth of **Pixmap** is one.

x, y Specifies the *x* and *y* coordinates, which indicate the hotspot. This hotspot is relative to the origin of the marker and must be a point within the marker.

XSetPolyMarker sets the marker in the specified graphics context. The hotspot is used to associate a point within the marker to a point within a window.

XSetPolyMarker can generate the errors **BadAlloc**, **BadGC**, **BadMatch**, and **BadPixmap**.

This routine is supported on the RT and the PS/2.

Note: On the PS/2, this routine is in **/usr/lib/liboldX.a**. Use **-loldX** on the command line to link to this library.

¦ Copyright IBM Corp. 1985, 1991

6.15.33 XStopAutoLoad

XStopAutoLoad(*display***)**
Display ******display*;

display Specifies the connection to the X Server.

XStopAutoLoad resets the **EventReport** mode of dial and lpfk devices.

This is supported on the RT only.

¦ Copyright IBM Corp. 1985, 1991

A.0 Appendix A. Fonts

This appendix describes a font support package for displays on RT systems and PS/2 systems. Topics discussed include:

Font file format used with the AIX RT X-Windows program **rtx** format)

Font file naming convention

Use of supplied font source files for use on the R

Use of a supplied program (METAFON (1)) to create and modify fonts for use on the RT

Use of a supplied facility to convert font files to **rtx** format for use on the RT

Use of a supplied facility to convert font files to **snf** font for use on the PS/2.

Note: The font information found under "bdftosnf" in topic A.16 applies to the PS/2.

(1) METAFONT is a trademark of Addison Wesley Publishing Company.

Subtopics

¦ Copyright IBM Corp. 1985, 1991

A.1 Overview of Font Support

The font package provided for RT display applications includes several fonts of various sizes and styles, as well as a program to create and modify fonts, particularly for use with various types of printers. A font that looks acceptable on a display device may be unsuitable for printed output because the resolution in pixels per inch of a printer is different from a display. On a printer, the resulting font size is generally too small to be usable.

METAFONT is a set of font compiler programs that allows you to design or modify your own 256-character fonts. These programs, font source files designed for use with these programs, and fonts used with RT displays are included with the font package. METAFONT is described in *The METAFONTbook*, by Donald E. Knuth (Addison Wesley Publishing Company, 1986). For details on developing various font styles and sizes with METAFONT, refer to Knuth's *Computer Modern Typefaces* (Addison Wesley Publishing Company, 1986). See "Using METAFONT to Create Fonts" in topic A.2 for details on using METAFONT in the X-Windows environment.

Before a font can be used with the RT displays, it must be in **rtx** format. In addition to providing a uniform bit storage scheme, the **rtx** format allows for fonts of variable widths and compression techniques. See "Converting Fonts to AIX RT X-Windows fonts" in topic A.9 for information on converting other font file formats to **rtx** format.

The supplied **rtx** font files, the METAFONT program, and the font source files can be optionally installed from a menu on the X-Windows diskettes. If you choose to install these fonts, they will be stored in **/usr/lpp/fonts**. Additional fonts that you create should also be stored here. The **rtx** font format is described in **/usr/include/rtfont.h**.

If you choose to install the font package, the METAFONT program and font source files for use with METAFONT fonts are installed. The supplied font source files, from which you can create new fonts with METAFONT, include only the characters from code page P0.

For a description of the significance of the fields in the font file names, see "Font File Naming Conventions."

Subtopics
A.1.1 Font File Naming Conventions

¦ Copyright IBM Corp. 1985, 1991

A.1.1 Font File Naming Conventions

This section describes the naming convention for **rtx** format font files. This convention is intended to associate a meaningful descriptor with a given file and to avoid storing duplicate fonts on the system.

Names for fonts supplied by IBM or created with METAFONT typically have these parts:

an alphabetic descriptor of the font styl

followed by a numeric value indicating the font point size by period,

a period for separation fro

the density factor which is related to output

The density factor is determined by multiplying a device's pixels-per-inch density times a scaling factor of 5.

For example, the **Rom10.500** font, supplied with the font package would be interpreted according to the following conventions:

Rom A roman font designed specifically for the RT displays. The initial capital letter in the font file name indicates that these fonts have been tuned for legibility on RT display devices. Contrast these file names with the untuned font source files listed in "Using METAFONT to Create Fonts" in topic A.2. The supplied fonts are divided into four types. They are:

- **Rom** Roman type, designed for clarity on RT displays.
- **Itl** Italic Roman type, designed for clarity on RT displays.
- **Bld** Boldface Roman type, designed for clarity on RT displays.
- **Erg** A sans-serif type, designed for clarity on RT displays.

You can develop your own alphabetic descriptors for font styles that you create with METAFONT. Note that font files provided with METAFONT and fonts described in Knuth's *Computer Modern Typefaces* use other descriptors, such as **cmr** (for Computer Modern Roman, a specific type of Roman font) and **cmrs** (for Computer Modern Roman slanted, a Roman font with some curve to the characters, but not as pronounced as italic).

If you are creating many fonts for use with the RT displays, you will probably create your own or adopt Knuth's conventions for alphabetic descriptors. Try to be consistent when you convert the font files to **rtx** format.

10 The font is displayed at 10 points (using the standard of 72 points per inch) on a device with the specified density factor, in this case 500.

¦ Copyright IBM Corp. 1985, 1991

. (period) Separates the point size from the density factor.

500 Determined by multiplying the scale factor five times the density of the device in pixels-per-inch, in this case 100.

Therefore, the **Rom10.500** font in this example would produce 10-point Roman type on a device that provides 100 pixels per inch. The same font used with a device providing 200 pixels per inch would appear half as large.

The standard of device pixels-per-inch density is established by the IBM 6155 Extended Monochrome Display (APA-16), which has a density of approximately 100 pixels per inch. The density factor for a 6155 display is 500 (5 times 100 pixels per inch). Therefore, on a 6155 display, font file **Rom6.500** produces 6-point Roman characters on the RT display; font file **cmr10.500**, an untuned font produced by METAFONT, produces 10-point Computer Modern Roman characters.

Before a font can be used with the RT display, it must be converted to **rtx** format. At conversion time, you have the opportunity to rename the converted file to the **rtx** conventions. IBM recommends using the **rtx** conventions for consistency among all font files so that you can more easily identify the point size resulting from a given font on a given device. Using the **rtx** convention also avoids duplication of fonts on the system.

¦ Copyright IBM Corp. 1985, 1991

A.2 Using METAFONT to Create Fonts

The following section describes commands used to create and modify METAFONT fonts. When installed, the METAFONT program resides in **/usr/lpp/mf**.

The METAFONT port provides several fonts and base files from which additional fonts can be created. For examples of the advanced font styles and characteristics that can be created with this font compiler program, see *Computer Modern Typefaces*. This book contains dozens of unique typefaces created with the METAFONT program, as well as some of the syntax statements that cause the typefaces to be produced.

In addition to the files provided with METAFONT, other font source files, which have already been built with METAFONT, are included.

The font source files represent the characters of code page P0 as implemented by the RT display. See *Keyboard Description and Character Reference* for a description of the P0, P1, and P2 characters.

You can copy these files, rename them, and build new characters or font styles into the copies. However, you must be relatively familiar with the techniques described in *The METAFONTbook* and *Computer Modern Typefaces*.

The font source files are installed in **/usr/lpp/mf/macros**.

The font source filenames do not begin with an uppercase letter, indicating that they have not been tuned for the RT display. The alphabetic descriptors (**rom, itl, bld**) and point sizes are consistent with the naming convention. The P0 stands for code page P0 to differentiate these files from the font files that include characters for P0, P1, and P2. The **mf** suffix identifies these files as METAFONT base files.

To create a font suitable for use on X-Windows from font source file **cmr10.mf**, perform the following:

1. Issue the **makefont** command.

   ```
   makefont -d cmr10
   ```

   This command creates a font for display only in **pk** format. The section titled "makefont" in topic A.7 describes the command, flags, and the METAFONT **pk** format.

2. Issue the **pktortx** conversion command.

   ```
   pktortx cmr10.590pk cmr10.500
   ```

   This command creates an **rtx** format font, (untuned, as indicated by the initial lowercase character in the file name), which produces 10-point characters (approximately) on a 100 pixels-per-inch device. The resulting font can be viewed with **gftype**, tuned if necessary, and used with the X-Windows program.

   Many METAFONT font files actually consist of several **.mf** files. The combined size of the files may be too large to be compiled. A program to compile a list of large files is described in **README.mf** in **/usr/lpp/mf/doc**.

3. Issue the **fixrtx** command to ensure that all characters in the font

¦ Copyright IBM Corp. 1985, 1991

file have proportional spacing imbedded in the raster image and are a constant height:

```
fixrtx -h cmr10.500
```

This command converts variable-height characters in an **rtx** font file to constant height characters and builds proportional spacing into a character's raster image. For more information, see "fixrtx" in topic A.11.

The METAFONT commands are defined in the following section.

¦ Copyright IBM Corp. 1985, 1991

A.3 cmmf

Purpose
Produces a generic font (**gf**) output file from a METAFONT (**mf**) input file.

Syntax

Description
The **cmmf** command invokes the METAFONT program using the Computer Modern **METAFONT** base. This base contains all of the plain base, as well as extensions used by the fonts in the Computer Modern family.

If the *file* argument has no extension, the default extension **.mf** is assumed.

Note: The **cmmf** command is similar to the **mf** command ("mf" in topic A.8). Use **cmmf** rather than **mf** when generating fonts from the Computer Modern source files in **/usr/lpp/mf/macros**.

Output is to one or more files on the current directory. The output files have the same name as the primary input file, but will have one of the following extensions:

*.nnn***gf** Identifies the generic font (gf) output file, containing the raster data for each character in the file. The value *nnn* is the resolution of the font (in pixels per inch) multiplied by the METAFONT scale factor. For example, the normal resolution of printer fonts is 240 pixels per inch. For a font file called **test** produced with the **makefont** command with no magnification specified, the output of **cmmf** on **test** would be **test.240gf**; at magnification **magstep1** (1.2 times the normal size), the output of **cmmf** on **test** would be **test.288gf**. See "makefont" in topic A.7 for more details on the **makefont** command.

.tfm Identifies a TEX (2) font metric (**tfm**) file. This file is used by the typesetting program TEX when formatting characters from this font. For information on TEX, see *The TEXbook*, by Donald E. Knuth (Addison Wesley Publishing Company, 1986).

.log Identifies a file that contains a log of all the terminal input and output (including error messages) that occurs while METAFONT is running.

The resolution of the generated font depends on the mode setting that METAFONT uses. By default, this mode is proof, which is used for generating proof copies of characters at very high resolution 2602 pixels per inch. To generate fonts for use by actual devices, you must specify either **pageprinter** for the IBM 3812 Pageprinter or the IBM Quietwriter¦ printers, to get 240 pixels-per-inch fonts, **proprinter** for the IBM 4201 Proprinter to get 240 pixels-per-inch fonts, or **displays** to get 100 pixels-per-inch fonts. See Example 3 for details on generating fonts for actual devices.

Note: This command is provided for the RT only.

¦ Copyright IBM Corp. 1985, 1991

Examples

1. To run METAFONT on a source file called **cmr10**:

```
cmmf  cmr10
```

   The file name extension **.mf** is assumed. Files output by this example would be **cmr10.2602gf, cmr10.tfm** and **cmr10.log**. The resolution of the output is 2602 pixels per inch because the default proof mode is used.

2. To run METAFONT on **cmr10** while searching for input files from a private macro library using the default shell:

```
MFINPUTS=.:/u/myid/mylib
export MFINPUTS
cmmf  cmr10
```

   The same example as above, but using the C shell:

```
setenv  MFINPUTS  .:/u/myid/mylib
cmmf  cmr10
```

3. To run METAFONT on source file **cmr10** to produce a font for the IBM 3812 Pageprinter, specifying **pageprinter** mode instead of the default proof mode:

```
cmmf "\mode=pageprinter; input cmr10"
```

   The output file from this example would be **cmr10.240gf**. The quotes are required to differentiate the ;(semicolon) and the \ (backslash) from the same characters that have special meanings to the shell.

Files

/usr/lpp/mf/bases/cm.base
/usr/lpp/mf/macros

Related Information

In this book: "gftopk" in topic A.4, "gftype" in topic A.5, "makefont" in topic A.7, "mf" in topic A.8.

The METAFONTbook, by Donald E. Knuth.

Computer Modern Typefaces, by Donald E. Knuth.

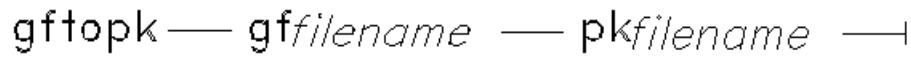
(2) TEX is a trademark of the American Mathematical Society.

¦ Copyright IBM Corp. 1985, 1991

A.4 gftopk

Purpose
Converts a generic font file (**gf**) to a packedfile (**pk**).

Syntax



Description
The **gftopk** command converts a generic font file (**gf***filename*) produced with the **cmmf** or **mf** commands to a packed font file (**pk***filename*). Both **gf***filename* and **pk***filename* must be specified. There are no default file names or extensions.

The packed font file that is output by **gftopk** is required to convert the METAFONT font to **rtx** format. By convention, the **pk** files produced for the Proprinter have a suffix of **Pk** (note the uppercase P) to distinguish them from the font files for the 3812 and Quietwriter (suffix **pk**). The distinction is necessary because the Proprinter has an aspect ratio of 3:5 rather than 1:1.

Note: This command is provided for the RT only.

Examples
To convert generic font file **cmr10.240gf** (Computer Modern Roman 10 point for a 240 pixels-per-inch device) to **pk** file **cmr10.1200pk**:

```
gftopk  cmr10.240gf  cmr10.1200pk
```

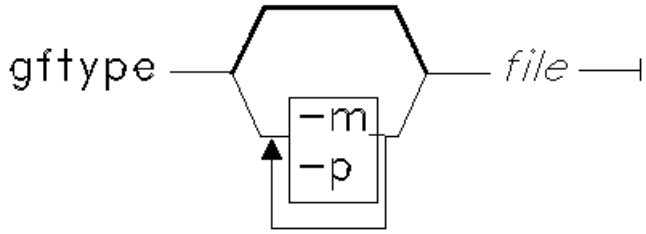
Related Information
In this book: "cmmf" in topic A.3, "makefont" in topic A.7, "mf" in topic A.8.

¦ Copyright IBM Corp. 1985, 1991

A.5 gftype

Purpose
Produces an ASCII dump of a METAFONT generic font file.

Syntax



Description
The **gftype** command takes a METAFONT **gf** file and produces an ASCII dump of its contents. The resulting dump contains a mnemonic representation of the data for each character, a pixel map of each character drawn as an array of asterisks and blanks, and the **tfm** data. Output is to **stdout** and can be redirected to a file by the normal AIX mechanisms.

Note: This command is provided for the RT only.

Flags

-m Deletes the per-character mnemonic information from the dump.

-p Deletes the per-character pixel maps from the dump.

Examples

1. To produce a complete dump, including mnemonics, pixel maps, and **tfm** data, of file **cmr10.240gf**:

   ```
   gftype cmr10.240gf
   ```

2. To produce a dump containing only the pixel maps and **tfm** data of file **cmr10.240gf**:

   ```
   gftype -m cmr10.240gf
   ```

3. To produce a complete dump, with the mnemonics and **tfm** data only, of file **cmr10.240gf**:

   ```
   gftype -p cmr10.240gf
   ```

Related Information
In this book: "cmmf" in topic A.3, "mf" in topic A.8.

The METAFONTbook, by Donald E. Knuth.

The TEXbook, by Donald E. Knuth.

¦ Copyright IBM Corp. 1985, 1991

A.6 inimf

Purpose
Produces a METAFONT base file to be loaded by the production version of METAFONT.

Syntax

```
inimf
```

See **Examples** for parameter data.

Description
The **inimf** command initializes the METAFONT program and loads the METAFONT internal tables with macro definitions, parameters, and other initialization values that make up a METAFONT base. The base you get depends on the specification of the command. Possible choices are plain base or Computer Modern base. If you do not specify all the parameters, the system will prompt you for parameters after you enter **inimf**.

The file that is output by **inimf** has an extension of **.base**. From a base file, you can run **cmmf** or **mf** to produce a **gf** font file.

Bases for system-wide use are stored on the METAFONT base library, **/usr/lpp/mf/bases**, but output from **inimf** goes to the current directory. Therefore, any base you create with **inimf** for use by other programs must be moved to the base library.

Note: This command is provided for the RT only.

Examples

Input to **inimf** can be specified interactively in response to prompts from METAFONT.

1. To create a plain METAFONT base file:

   ```
   inimf "plain; input local; dump"
   ```

   This sequence creates the files **plain.base** and **plain.log** on the current directory. The **local.mf** file is input just before dumping to define the printer and display modes of METAFONT. To test your new base, issue the **mf** command. METAFONT searches the current directory first for base files. See "mf" in topic A.8 for information on how to use the MFBASES environment variable to control the METAFONT search for base files.

2. To create the Computer Modern base, **cm.base**:

   ```
   inimf "plain; input cmbase; input local; dump"
   ```

   The **inimf** command creates its output on the file **plain.base**, since **plain** is the first file name **inimf** sees. Use the **mv** command to change **plain.base** to the proper name.

Files

¦ Copyright IBM Corp. 1985, 1991

/usr/lpp/mf/macros

Related Information
In this book: "cmmf" in topic A.3, "mf" in topic A.8.

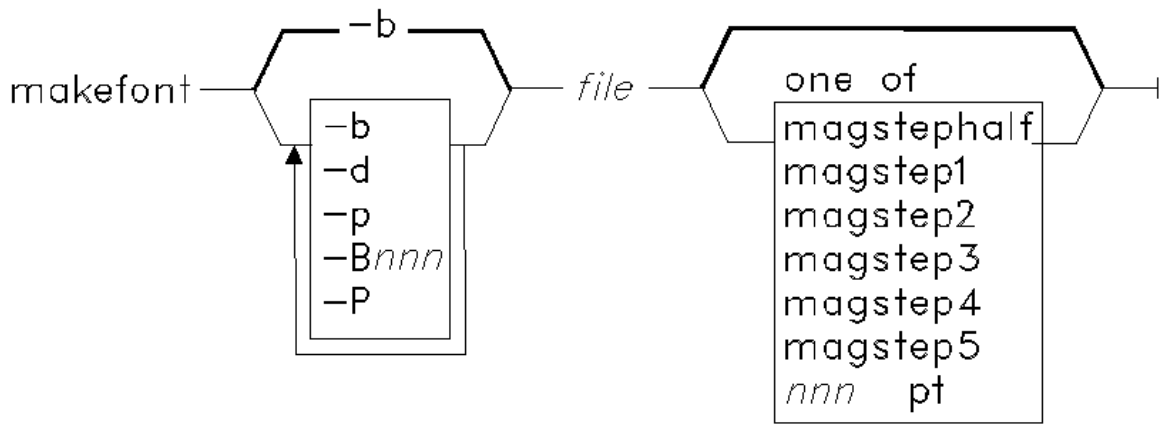
¦ Copyright IBM Corp. 1985, 1991

A.7 makefont

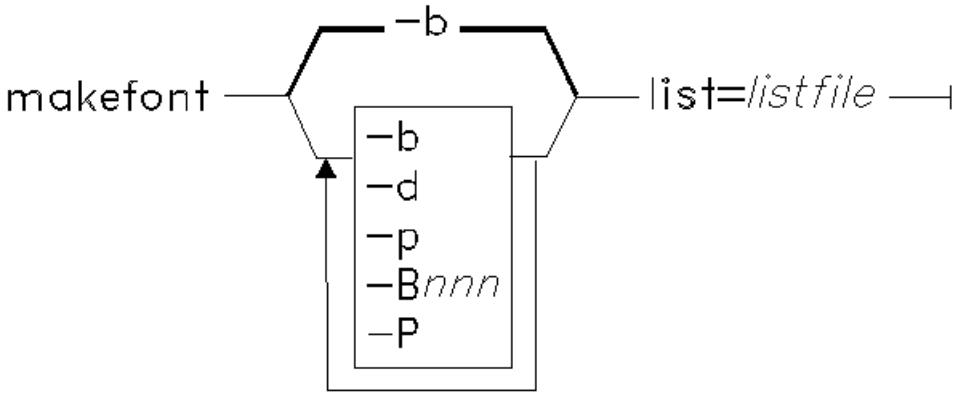
Purpose

Produces a display font and printer font from one or more METAFONT source files.

Syntax



-or-



Description

The **makefont** command generates Computer Modern fonts for use on displays or printers. METAFONT is invoked using the **cm** base. The **makefont** command calls **gftopk** to convert the generic font **gf** file to a packed font (**pk**) file. The input *file*, therefore, should be a Computer Modern source file without the **.gf** extension.

By default, this command generates fonts for both the printers 3812 and Quietwriter and the displays unless the **-d** or **-p** flags are specified.

The first form of the command generates output from only one font source file. You would use this form to generate a single font of a specific size. For example, suppose you need a font for printed output, but find that the font does not exist on the library in the size that you need. Use **makefont** with the desired magnification to create a font of the required size.

The second form of the command generates output from all the font source

¦ Copyright IBM Corp. 1985, 1991

files in *listfile*. Each line of *listfile* contains either a comment (the first character of the command line is **%**) or the operands of the first form of the **makefont** command. For example:

```
% Make a font for a printer only.
makefont cmr10 magstep3 -p
% Make a font for a display only.
makefont cmr10 8pt -d
% Make a font for a printer and display.
makefont cmr12 magstep1 -b
```

If you have write permission to **/usr/lpp/fonts**, you can store newly-created fonts directly to the font library. Otherwise, the fonts you create are stored in the current directory where you are working and must be moved to the font library for system-wide use.

Output to the font library consists of the **tfm** file for the font and one or more packed font files (**pk**). To avoid accumulating many files when running **makefont** on a long list of font names, METAFONT erases the **.log** file and the **gf** files (when the corresponding **pk** is created).

Output to the current directory includes the **gf**, **pk**, and **.log** files.

METAFONT input files are searched for on the current directory and then on the METAFONT macro library, **/usr/lpp/mf/macros**. You can specify your own search order by setting the MFINPUTS environment variable to a list of path names (separated by colons).

The METAFONT base file is searched for on the current directory and then on the METAFONT base library, **/usr/lpp/mf/bases**. This search order can be overridden by setting the MFBASES environment variable. If **makefont** determines that a particular font file already exists on the output library, either **/usr/lpp/fonts** or the current directory, the **makefont** bypasses METAFONT execution for that file. To replace a font file, you must first delete the appropriate **pk** file from the output library. If you find that you have a special font-generation requirement that **makefont** cannot handle, copy **/usr/bin/makefont** to one of your own directories and modify it.

Note: This command is provided for the RT only.

Flags

The first flag can be used to specify whether both printer and display fonts are produced and can also indicate the base from which to produce the fonts. The second flag can be used to specify the magnification of the resulting font. This is typically used when you have an existing font and simply need to resize it for a printer or a display.

The first set of flags are defined as follows:

-b Generate printer (3812 and Quietwriter only) and display fonts. This is the default.

-d Generate display fonts only.

-p Generate 3812 and Quietwriter fonts only.

¦ Copyright IBM Corp. 1985, 1991

-B*nnn* Name of the base file to be loaded at the start of METAFONT execution. By default, the **cm** base is used.

-P Generate Proprinter fonts only. To generate all three types of printer fonts, specify the flags **-P -b**, in that order.

The magnification flags are defined as follows:

magstephalf 1.095 times the size of the input font file.

magstep1 1.2 times the size of the input font file.

magstep2 1.44 times the size of the input font file.

magstep3 1.728 times the size of the input font file.

magstep4 2.074 times the size of the input font file.

magstep5 2.488 times the size of the input font file.

Note: The **magstep** flags can be used with any font.

*nnn***pt** Size is *nnn* points.

This flag works only if *file* ends in a decimal number specifying the design size of the font (such as **cmr10**). The **makefont** command reads this numeric suffix and uses it with the value of *nnn* to create a fractional magnification which it passes to METAFONT. For example, specifying **cmr10 15pt** passes METAFONT a magnification value of 15/10.

Examples

1. To create both printer and display fonts for the **cmr10** font (normal size):

```
makefont  cmr10
```

   This example creates **cmr10.tfm**, **cmr10.590pk** (the display font), and **cmr10.1200pk** (the printer font).

2. To create the same font at magnification **magstep4**:

```
makefont  cmr10 magstep4
```

   This example creates **cmr10.tfm**, **cmr10.1223pk** (the display font), and **cmr10.2488pk** (the printer font).

3. To generate a display font only from **testfont.mf**:

```
makefont -d  testfont
```

4. To generate display and printer fonts for a font of your own design, using your own **mybase.base** file:

```
makefont  -Bmybase myfont
```

Files

/usr/bin/makefont

¦ Copyright IBM Corp. 1985, 1991

/usr/lpp/mf/bases/cm.base
/usr/lpp/mf/bases/plain.base
/usr/lpp/mf/macros

Related Information
In this book: "cmmf" in topic A.3, "gftopk" in topic A.4, "mf" in topic A.8.

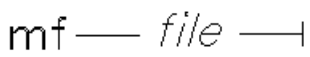
The METAFONTbook, by Donald E. Knuth.

¦ Copyright IBM Corp. 1985, 1991

A.8 mf

Purpose
Produces a generic font (**gf**) output file from a METAFONT (**mf**) input file.

Syntax



Description
The **mf** command invokes the METAFONT program using the plain **METAFONT** base.

If the *file* argument has no extension, the default extension **.mf** is assumed.

Note: The **mf** command is very similar to the **cmmf** command ("cmmf" in topic A.3). The **mf** command generates fonts from the plain source files on **/usr/lpp/mf/macros**; **cmmf** generates fonts from the Computer Modern source files.

Output is to one or more files on the current directory. The output files have the same name as the primary input file, but will have one of the following extensions:

*.nnn***gf** Identifies the generic font (gf) output file, containing the raster data for each character in the file. The value *nnn* is the resolution of the font (in pixels per inch) multiplied by the METAFONT scale factor. For example, the normal resolution of printer fonts is 240 pixels per inch. For a font file called **test** produced with the **makefont** command with no magnification specified, the output of **mf** on **test** would be **test.240gf**; at magnification **magstep1** (1.2 times the normal size), the output of **mf** on **test** would be **test.288gf**. See "makefont" in topic A.7 for more details on the **makefont** command.

.tfm Identifies a TEX font metric file (**tfm**). This file is used by the typesetting program TEX when formatting characters from this font. For information on TEX, see *The TEXbook*, by Donald E. Knuth (Addison Wesley Publishing Company, 1986).

.log Identifies a file that contains a log of all the terminal input and output (including error messages) that occurs while METAFONT is running.

The resolution of the generated font depends on the mode setting that METAFONT uses. By default, this mode is proof, which is used for generating proof copies of characters at very high resolution (2602 pixels per inch). To generate fonts for use by actual devices, you must specify a mode of either **pageprinter** to get 240 pixels-per-inch fonts, **proprinter** to get 240 pixels-per-inch fonts, or **displays** to get 100 pixels-per-inch fonts. See Example 3 for details on generating fonts for actual devices.

Note: This command is provided for the RT only.

Examples

1. To run METAFONT on a source file called **sample.mf**:

¦ Copyright IBM Corp. 1985, 1991

```
mf  sample
```

The filename extension **.mf** is assumed. Output is to **sample.2602gf** and **sample.log**. The resolution of the output is 2602 pixels per inch because the default **proof** mode is used.

2. To run METAFONT on **cmr10** while searching for input files from a private macro library using the default shell:

```
MFINPUTS=.:/u/myid/mylib
export MFINPUTS
cmmf  cmr10
```

The same example as above, but using the C shell:

```
setenv  MFINPUTS  .:/u/myid/mylib
cmmf  cmr10
```

3. To run METAFONT on source file **cmr10** to produce a font for the IBM 3812 Pageprinter, specifying **pageprinter** mode instead of the default proof mode:

```
cmmf "\mode=pageprinter; input cmr10"
```

The output file from this example would be **cmr10.240gf**. The quotes are required to differentiate the ;(semicolon) and the \ (backslash) from the same characters that have special meanings to the shell.

Files

/usr/lpp/mf/bases/plain.base
/usr/lpp/mf/macros

Related Information
In this book: "cmmf" in topic A.3, "gftopk" in topic A.4, "gftype" in topic A.5, "makefont" in topic A.7.

The METAFONTbook, by Donald E. Knuth.

¦ Copyright IBM Corp. 1985, 1991

A.9 Converting Fonts to AIX RT X-Windows fonts

Before a font can be used on the RT display, it must be stored in **rtx** format.

The commands described in this section can be used to perform the following font-conversion operations:

Convert a **bdf** (bitmap distribution format) font to an **rtx** font.

Convert variable-height characters in an **rtx** font file to constant-height characters

Creates a **fonts.dir** file from directory of font files.

Convert a METAFONT packed font **pk**) file to an X-Windows **rtx** font file.

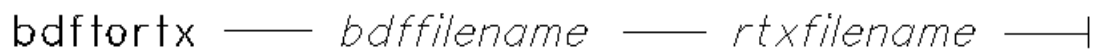
Convert a **snf** (server-natural format) font to an **rtx** font.

¦ Copyright IBM Corp. 1985, 1991

A.10 bdftortx

Purpose
Converts a BDF (bitmap distribution format) font file to the **rtx** font.

Syntax



Description
The **bdftortx** command converts a font in the Character Bitmap Distribution Format (**bdf**) to the X-Windows **rtx** font file format. This command takes a **bdf** font file as input. The result is a file in **rtx** format that can be used with RT displays.

Note: This command is provided for the RT only.

Examples
To convert the **bdf** font file **fixed.bdf** to an **rtx** font file:

```
bdftortx  fixed.bdf  /usr/lpp/fonts/fixed
```

Files

/usr/lpp/fonts

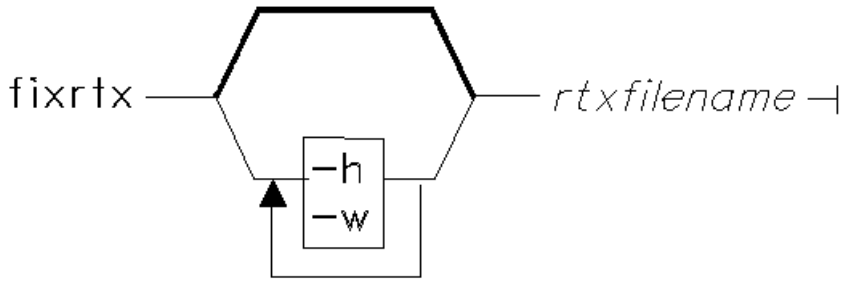
¦ Copyright IBM Corp. 1985, 1991

A.11 fixrtx

Purpose

Converts an **rtx** font file that contains proportional spacing parameters into a file with the proportional spacing built into the raster image.

Syntax



Description

For **rtx** font files to work in the X-Windows environment, any proportional spacing (blank or white space preceding or following the character image) must be contained in the raster image. In addition, all the characters in the font file must be a constant height. (Variable-height characters within a single font file are not allowed.) The **fixrtx** command takes an **rtx** font file and builds any proportional spacing into the raster image. With the **-h** flag of the command, all the characters in the font file are made a constant height. The constant height is determined by the tallest character in the *rtxfilename* file.

This command will not alter the file if it is already in the correct format. You must execute this command with the **-h** flag specified on **rtx** files created with the **pktortx** command if they are intended to be used in the X-Windows environment.

The name of the **rtx** font file does not change.

Note: This command is provided for the RT only.

Flags

-h Change all characters to a constant height equal to the height of the tallest character in the font. Specify this flag for all font files that are intended to be used in the X-Windows environment.

-w Change all characters to a constant width equal to the width of the widest character in the font.

Examples

To convert **rtx** font file **itl10.500** (which includes characters of variable heights) to a file of constant-height characters:

```
fixrtx -h itl10.500
```

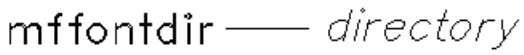
The resulting file retains the name of the input file, but the characters are all a constant height and any proportional spacing is built into the character's raster image.

¦ Copyright IBM Corp. 1985, 1991

A.12 mkfontdir

Purpose
Creates a **fonts.dir** file from directory of font files.

Syntax



Description
For each directory argument, **mkfontdir** reads the **rtx** files in the directory. It reads the *font_id* field in the **rtx** *font_header* structure. Use **mkfontdir** each time a new font is added to the font directory.

The *font_ids* and the associated filenames are written into the **fonts.dir** file in the argument directory.

A **fonts.alias** file is used to map new names to existing font names. This file can be edited with an ASCII editor. It can be placed in any font directory.

The **fonts.alias** file contains two columns of font names. The first column lists the aliases and the second column lists the *font_name* patterns.

When a font alias is used, it references a name in a directory. Each directory in the font search path is searched for the font name. Alias font names are not required to use fonts in the same directory as the alias file.

To embed white-space in the name or the alias, enclose the name or the alias with double quotes (") preceeded with a backslash (\). (Any imbedded characters should be preceeded with a backslash.)

Note: This command is provided for the RT only.

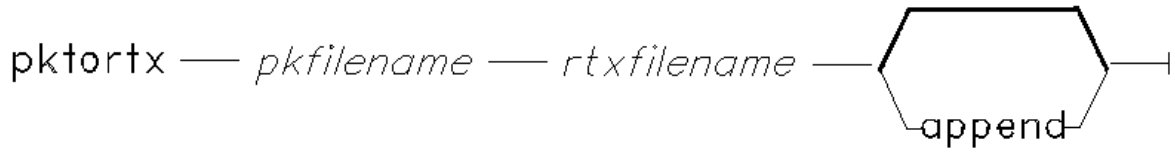
Files
/usr/lpp/fonts

¦ Copyright IBM Corp. 1985, 1991

A.13 pktortx

Purpose
Converts a METAFONT **pk** font file to the RT display **rtx** font file format.

Syntax



Description
Before any of the standard fonts provided with or produced by the METAFONT program can be used with the X-Windows program, they must be converted to the X-Windows **rtx** font file format. This command takes as input a **pk** METAFONT file (note that this conversion does not work for **gf** type files). The resulting output file is in **rtx** format and can be used with an RT display if all the characters in the font file are of a constant height.

Note that **pk** font files typically contain characters of variable widths and variable heights. In this case, the **rtx** font file produced by **pktortx** will also contain characters of variable widths and heights. Therefore, before the **rtx** font file generated by this command can be used in the X-Windows environment, the characters in the file must be converted to a constant height. See "fixrtx" in topic A.11 for instructions on this conversion command.

Note: This command is provided for the RT only.

Flags
The **append** option puts multiple **pk** format files into a single **rtx** file. This option is required if the original source file is too large to be compiled with the **makefont** command. In this case, the source file must be broken down into multiple files. Each file is first compiled into several **pk** files using the **makefont** command. Each **pk** file that was required to contain all the information in the original source file is then appended to *rtxfilename* in **rtx** format.

For example, if a METAFONT source file requires three separate **mf** files to compile and, therefore, requires three separate **pk** files), the **pktortx** command is first issued to establish the name of the **rtx** file. The command is then issued with the other **pk** file names, the same **rtx** file name, and the **append** option.

```
pktortx /mydir/pkfile1 /mydir/rtxfile
pktortx /mydir/pkfile2 /mydir/rtxfile append
pktortx /mydir/pkfile3 /mydir/rtxfile append
```

This example results in one **rtx** font file, **/mydir/rtxfile**. The user must ensure that each **pk** file to be appended was created using the same parameter files and that each file has unique character codes.

Examples
To convert the **pk** font file **/usr/lpp/fonts/cmr10.1200pk** to an **rtx** font file:

¦ Copyright IBM Corp. 1985, 1991

```
pktortx  /usr/lpp/fonts/cmr10.1200pk  /usr/lpp/fonts/cmr10.1200
fixrtx -h  /usr/lpp/fonts/cmr10.1200
```

Note how the output file name conforms to the **rtx** naming convention with an expanded alphabetic descriptor classification for Computer Modern Roman.

Files
/usr/lpp/fonts

Subtopics
A.13.1 Related Information

¦ Copyright IBM Corp. 1985, 1991

A.13.1 Related Information

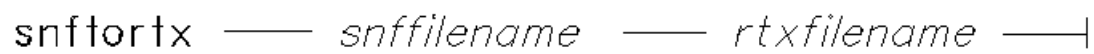
In this book: "fixrtx" in topic A.11.

¦ Copyright IBM Corp. 1985, 1991

A.14 snftortx

Purpose
Converts an SNF (server-natural font) font file to the RT display **rtx** font.

Syntax



Description
The **snftortx** command converts a font in Server Natural Font Format (**snf**) to the **rtx** format. Before any of the fonts in **snf** can be used with X-Windows, they must be converted to the X-Windows **rtx** font file format.

Note: This command is provided for the RT only.

Examples
To convert the **snf** font file **fixed.snf** to an **rtx** font file:

```
snftortx  fixed.snf  /usr/lpp/fonts/fixed
```

Files

/usr/lpp/fonts

¦ Copyright IBM Corp. 1985, 1991

A.15 Converting Fonts to AIX PS/2 X-Windows Fonts

The AIX PS/2 X-Windows licensed program uses **snf** fonts. Since **snf** fonts are optimized for a specific system and display, the same **snf** fonts cannot be used for both RT and PS/2 systems. To facilitate font distribution, a portable font format, called **bdf** is used. The **bdftosnf** command, which is described in this section, converts these portable **bdf** (bitmap distribution format) files to **snf** (server natural format) files specifically for the PS/2.

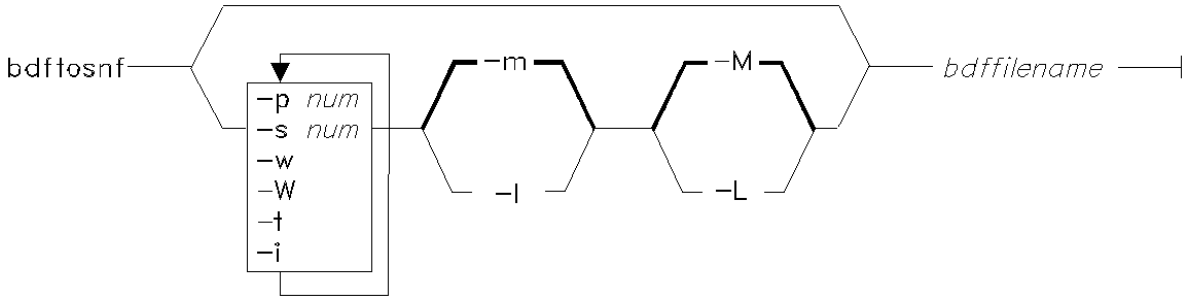
¦ Copyright IBM Corp. 1985, 1991

A.16 bdftosnf

Purpose

Converts a BDF (bitmap distribution font) font file to a PS/2-specific **snf** font.

Syntax



Description

The **bdftosnf** command converts a font in Bitmap Distribution Format (**bdf**) to Server-Natural Format (**snf**) for the PS/2. **bdftosnf** takes the file specified by *bdffilename*, or standard input if no file name is specified, and writes a server-natural font to standard output, which can be redirected to a file.

The AIX PS/2 X-Windows licensed program uses **snf** fonts. Since these **snf** fonts are optimized for a specific system and display, the same **snf** fonts cannot be used for both RT and PS/2 systems. To facilitate font distribution, a portable font format, called **bdf** is used. The **bdftosnf** command converts these portable **bdf** files to **snf** files specifically for the PS/2.

Note: This command is provided for the PS/2 only.

Flags

-i Converts terminal-emulator fonts without computing correct ink metrics for them.

-l Sets the bit order to be least significant bit first.

-L Sets the byte order to be least significant bit first.

-m Sets the bit order to be most significant bit first.

-M Sets the byte order to be most significant bit first.

-p *num* Sets the glyph padding to the specified value of *num*. Possible values are: **1**, **2**, **4**, or **8**.

-s *num* Sets the scanline unit padding to the specified value of *num*. Possible values are: **1**, **2**, **4**, or **8**.

-t Expands glyphs in terminal-emulator fonts to fill the bounding box.

-w Prints warnings if the character bitmaps have bits set to outside of their defined widths.

¦ Copyright IBM Corp. 1985, 1991

-W Prints warnings for characters with an encoding of -1. If this flag is not specified, these characters are ignored with no warning.

Examples

To convert the **bdf** font file **fixed.bdf** to a PS/2 **snf** font file:

```
bdftosnf  fixed.bdf  >fixed.snf
```

¦ Copyright IBM Corp. 1985, 1991

B.0 Appendix B. Xlib Cursor Fonts

The following is a list of cursors and their shapes in fonts that are currently available on X-Windows. These cursors are defined in **/usr/include/Xll/cursorfont.h**.

#define	**XC_num_glyphs**	154		
#define	**XC_X_cursor**	0	#define	**XC_**
#define	**XC_based_arrow_dow4**		#define	**XC_**
#define	**XC_boat**	8	#define	**XC_**
#define	**XC_bottom_left_cor12r**		#define	**XC_**
#define	**XC_bottom_side**	16	#define	**XC_**
#define	**XC_box_sprial**	20	#define	**XC_**
#define	**XC_circle**	24	#define	**XC_**
#define	**XC_coffee_mug**	28	#define	**XC_**
#define	**XC_cross_reverse**	32	#define	**XC_**
#define	**XC_diamond_cross**	36	#define	**XC_**
#define	**XC_dot_box_mask**	40	#define	**XC_**
#define	**XC_draft_large**	44	#define	**XC_**
#define	**XC_draped_box**	48	#define	**XC_**
#define	**XC_fleur**	52	#define	**XC_**
#define	**XC_gumby**	56	#define	**XC_**
#define	**XC_hand1_mask**	60	#define	**XC_**
#define	**XC_icon**	64	#define	**XC_**
#define	**XC_left_ptr**	68	#define	**XC_**
#define	**XC_left_tee**	72	#define	**XC_**
#define	**XC_ll_angle**	76	#define	**XC_**
#define	**XC_man**	80	#define	**XC_**
#define	**XC_mouse**	84	#define	**XC_**
#define	**XC_pirate**	88	#define	**XC_**
#define	**XC_question_arrow**	92	#define	**XC_**
#define	**XC_right_side**	96	#define	**XC_**
#define	**XC_rightbutton**	100	#define	**XC_**
#define	**XC_sailboat**	104	#define	**XC_**
#define	**XC_sb_h_double_arr10**8		#define	**XC_**
#define	**XC_sb_right_arrow**	112	#define	**XC_**
#define	**XC_sb_v_double_arr11**6		#define	**XC_**
#define	**XC_sizing**	120	#define	**XC_**
#define	**XC_spraycan**	124	#define	**XC_**
#define	**XC_target**	128	#define	**XC_**
#define	**XC_top_left_arrow**	132	#define	**XC_**
#define	**XC_top_right_corne**136		#define	**XC_**
#define	**XC_top_tee**	140	#define	**XC_**
#define	**XC_ul_angle**	144	#define	**XC_**
#define	**XC_ur_angle**	148	#define	**XC_**
#define	**XC_xterm**	152	#define	**XC_**

¦ Copyright IBM Corp. 1985, 1991

C.0 Appendix C. HFT Functions and Datastream Support

Subtopics
C.1 aixterm HFT Functions
C.2 aixterm Datastream Support

¦ Copyright IBM Corp. 1985, 1991

C.1 aixterm HFT Functions

VTLs are supported with the following characteristics:

Locator reports are given for the following mouse events

- **ButtonPress**
- **ButtonRelease**
- **MotionNotify** (with any button down)

Locator reports are given in absolute coordinates

Locator motion is processed in compressed mode

Applications need not display a locator cursor

A font table is maintained for each invocation of **aixterm** to support the change font VTD.

The file **/usr/lpp/X11/defaults/Xfonts** identifies the fonts configured on the RT. This file controls the mapping of font IDs to font files recognized by **aixterm**. A request from a program to **aixterm** to query font VTD returns the font ID based on the contents of the **Xfonts** file. The change font VTD changes the local **aixterm** font table according to the IDs defined in the **Xfonts** file.

The format of each line in the **Xfonts** file is:

id path style attr width height

The lines in the file that start with # are ignored. The fields within a line must be separated by one or more blanks.

The parts of each line in **Xfonts** indicate the following:

id	Specifies the ID to be associated with a particular font file. The decimal values 0 through 65536 are valid. Other programs can add to this file once their fonts match the common font file format.
path	Specifies the name of the font file to be opened. The font file must be in the directory **/usr/lpp/fonts**.
style	Specifies a number identifying the style of the font.
attr	Specifies a number identifying the attributes of the font. The defined attributes are: **0** (HFFNTPLAIN) **1** (HFFNTBOLD) **2** (HFFNTITALIC) Refer to the file **/usr/include/sys/hft.h** for definitions of HFFNTPLAIN, HFFNTBOLD, and HFFNTITALIC.
width	Specifies the width of the font characters in pels.
height	Specifies the height of the font characters in pels.

Copyright IBM Corp. 1985, 1991

For more information, see "hft" and "remote processing" in the *AIX Operating System Technical Reference*.

Function	**HFT**	**aixterm**
TERMINAL CONTROL		
Query I/O Error	ioctl	no support
Query Device	ioctl	no support
Reconfigure	ioctl	no support
Get Channel Number	ioctl	no support
Set Echo and Break Map	ioctl	no support
Set Keyboard Map	ioctl	hftctl
Get Virtual Terminal ID	ioctl	no support
Query Device ID	ioctl	no support
Query Physical Device (ID=0)	ioctl	hftctl
Query HFT device	ioctl	hftctl
Query Locator	ioctl	hftctl
Query LPFKs	ioctl	no support
Query Dials	ioctl	no support
Query PS	ioctl	hftctl
Enable / Disable Sound	ioctl	no support
Enter / Exit Monitor Md	ioctl	no support
Query Screen Manager	ioctl	no support
Control Screen Manager	ioctl	no support
KSR Protocol	VTD	VTD
Character Set Definition	VTD	no support
Set KSR Color Palette	VTD	VTD
Change Fonts	VTD	VTD
Cursor Representation	VTD	VTD
INPUT		
Dead Key Support	read	supported
Code Page Shift	read	supported

Copyright IBM Corp. 1985, 1991

Untranslated Key	read	supported
Input Device Report	read	supported
mouse	VTL	supported
tablet	VTL	no support
dials	VTL	no support
Lighted Program Function Keys	VTL	no support
Sound Complete	signal	no support
OUTPUT		
Write ASCII (Code Page 850)	write	supported
Write Code Page Shift	write	supported
Set LEDs	VTD	no support
Set Locator Threshold	VTD	no support
Set Table Dead Zone	VTD	no support
Set LPFKs	VTD	no support
Set Dials	VTD	no support
Sound output	VTD	no support
Cancel Sound	VTD	no support
Change Physical Display	VTD	no support

¦ Copyright IBM Corp. 1985, 1991

C.2 aixterm Datastream Support

The following is a list of the escape sequences supported by **aixterm**.

Some escape sequences activate and deactivate an alternate screen buffer that is the same size as the display area of the window. This capability allows the contents of the screen to be saved and restored. When the alternate screen is activated, the current screen is saved and replaced with the alternate screen. The saving of lines scrolled off of the window is disabled until the original screen is restored.

This table uses these abbreviations in the righthand column:

Xv Supported by aixterm running in vt100 mode.
Xh Supported by aixterm running in hft mode.
H Found in hft datastream.
V Found in vt100 datastream.

Name	**Function**	**Datastream**	**Support**
	SINGLE-BYTE CONTROLS		
BEL	Bell	0x07	Xv, Xh, H, V
BS	Backspace	0x08	Xv, Xh, H, V
HT	Horizontal tab	0x09	Xv, Xh, H, V
LF	Linefeed	0x0A	Xv, Xh, H, V
VT	Vertical tab	0x0B	Xv, Xh, H, V
FF	Form feed	0x0C	Xv, Xh, H, V
CR	Carriage return	0x0D	Xv, Xh, H, V
SO	Shift out	0x0E	Xv, Xh, H, V
SI	Shift in	0x0F	Xv, Xh, H, V
DC1	Device control 1	0x11	H, V
DC3	Device control 3	0x13	H, V
CAN	Cancel	0x18	H, V
SUB	Substitute (also cancels)	0x1A	H, V
ESC	Escape	0x1B	Xv, Xh,

Copyright IBM Corp. 1985, 1991

			H, V
SS4	Single Shift 4	0x1C	Xh, H
SS3	Single Shift 3	0x1D	Xh, H
SS2	Single Shift 2	0x1E	Xh, H
SS1	Single Shift 1	0x1F	Xh, H
cbt	cursor back tab	ESC [Pn Z	Xv, Xh, H
cha	cursor horizontal absolute	ESC [Pn G	Xv, Xh, H
cht	cursor horizontal tab	ESC [Pn I	H
ctc	cursor tab stop control	ESC [Pn W	H
cnl	cursor next line	ESC [Pn E	H
cpl	cursor preceding line	ESC [Pn F	Xv, Xh, H
cpr	cursor position report	ESC [Pl; Pc R	Xv, Xh, H, V
cub	cursor backward	ESC [Pn D	Xv, Xh, H, V
cud	cursor down	ESC [Pn B	Xv, Xh, H, V
cuf	cursor forward	ESC [Pn C	Xv, Xh, H, V
cup	cursor position	ESC [Pl; Pc H	Xv, Xh, H, V
cuu	cursor up	ESC [Pn A	Xv, Xh, H, V
cvt	cursor vertical tab	ESC [Pn Y	H
da1	DEVICE ATTRIBUTES request (host to vt100) request (host to vt100) response (vt100 to host)	 ESC [c ESC [0 c ESC [? 1 ; 2 c	 Xv, Xh, V Xv, Xh, V Xv, Xh, V
dch	delete character	ESC [Pn P	Xv, Xh, H
decaln	screen alignment display	ESC # 8	Xv, Xh, V
deckpam	keypad application mode	ESC =	Xv, V
deckpnm	keypad numeric mode	ESC >	Xv, V
decrc	restore cursor & attributes	ESC 8	Xv, Xh, V
decsc	save cursor & attributes	ESC 7	Xv, Xh, V

Copyright IBM Corp. 1985, 1991

decstbm	set top & bottom margins	ESC [Pt; Pb r	Xv, Xh, V
dl	delete line	ESC [Pn M	Xv, Xh, H
dsr	device status report 0 response from vt100: ready 5 command from host: please report status 6 command from host: report active position 13 error report sent from virtual terminal to host	ESC [Ps n	 Xv, Xh, V Xv, Xh, V Xv, Xh, H, V H
dmi	disable manual input	ESC ` (back quote)	H
emi	enable manual input	ESC b	H
ea	erase area 0 erase to end of area 1 erase from area start 2 erase all of area	ESC [Ps O	 Xv, Xh, H Xv, Xh, H Xv, Xh, H
ed	erase display 0 erase to end of display 1 erase from display star 2 erase all of display	ESC [Ps J	 Xv, Xh, H, V Xv, Xh, H, V Xv, Xh, H, V
ef	erase field-e,s,all 0 erase to end of field 1 erase from field start 2 erase all of field	ESC [Ps N	 Xv, Xh, H Xv, Xh, H Xv, Xh, H
el	erase line 0 erase to end of line 1 erase from start of line 2 erase all of line	ESC [Ps K	 Xv, Xh, H, V Xv, Xh, H, V Xv, Xh, H, V
ech	erase character	ESC [Pn X	Xv, Xh, H
gsm	graphic size modify	ESC [Pn; Pn Sp B	H
hts	horizontal tab stop	ESC H	Xv, Xh, H, V
hvp	horizontal and vertical position	ESC [Pl; Pc f	Xv, Xh, H, V
ich	insert character	ESC [Pn @	Xv, Xh, H
il	insert line	ESC [Pn L	Xv, Xh, H

Copyright IBM Corp. 1985, 1991

ind	index	ESC D	Xv, Xh, H, V
ls2	lock shift G2	ESC n	Xv
ls3	lock shift G3	ESC o	Xv
nel	next line	ESC E	Xv, Xh, H, V
ksi	keyboard status information	ESC [Ps p	H
pfk	PF key report	ESC [Pn q	Xh, H
rcp	restore cursor position	ESC [u	Xv, Xh, H
ri	reverse index	ESC M	Xv, Xh, H, V
ris	reset to initial state	ESC c	Xv, Xh, H, V
rm	reset mode, ANSI specified modes: See "set mode" below in this column.	ESC [Ps;...;Ps l	
	reset mode, other private modes and XTERM private modes: See "set mode" below in this column.	ESC [? Ps;...;Ps l	
	restore mode, other private modes and XTERM private modes: See "set mode" below in this column.	ESC [? Ps;...;Ps r	
	save mode, other private modes and XTERM private modes: See "set mode" below in this column.	ESC [? Ps;...;Ps s	
scp	save cursor position	ESC [s	Xv, Xh, H
scs	select character set		
	United Kingdom Set	ESC (A (G0) ESC) A (G1) ESC * A (G2) ESC + A (G3)	Xv, V Xv, V Xv, V Xv, V
	ASCII Set (USASCII)	ESC (B (G0) ESC) B (G1) ESC * B (G2) ESC + B (G3)	Xv, V Xv, V Xv, V Xv, V

Copyright IBM Corp. 1985, 1991

	special graphics	ESC (0 (G0) ESC) 0 (G1) ESC * 0 (G2) ESC + 0 (G3)	Xv, V Xv, V Xv, V Xv, V
sd	scroll down	ESC [Pn T	H
sl	scroll left	ESC [Pn Sp @	H
sr	scroll right	ESC [Pn Sp A	H
ss2	single shift G2	ESC N	Xv
ss3	single shift G3	ESC O	Xv
su	scroll up	ESC [Pn S	Xv, Xh, H
sgr	set graphic rendition 0 normal 1 bold 4 underscore 5 blink (appears as bold) 7 reverse 8 invisible 10..17 fonts 30..37 foreground colors 40..47 background colors 90..97 foreground colors 100..107 background colors	ESC [Ps m	 Xv, Xh, H, V Xv, Xh, H, V Xv, Xh, H, V Xv, Xh, H, V Xv, Xh, H, V Xh, H Xh, H Xh, H Xh, H Xh, H Xh, H
sg0a	set GO character set	ESC (f	Xh, H
sg0b	set GO character set-ALT	ESC , f	H
sg1a	set G1 character set	ESC) f	Xh, H
sg1b	set G1 character set-ALT, where f := {:;<=>?@}	ESC - f	H
sm	set mode		
	ANSI specified modes 4 IRM insert mode 12 SRM send/rec mode 18 TSM tab stop mode 20 LNM linefeed/newline	ESC [Ps;...;Ps h	 Xv, Xh, H H H Xv, Xh, H, V

Copyright IBM Corp. 1985, 1991

	Other private modes 1 normal/application cursor 3 80/132 columns 4 smooth/jump scroll 5 reverse/normal video 6 origin/normal 7 on/off autowrap 8 on/off autorept 21 CNM CR-NL	ESC [? Ps;...;Ps h	 Xv, V Xv, Xh, V Xv, Xh, V Xv, Xh, V Xv, Xh, V Xv, Xh, H, V Xv, Xh, V H
	XTERM private modes 40 132/80 column mode 41 curses(5) fix 42 hide/show scrollbar 43 on/off save scroll text 44 on/off margin bell 45 on/off reverse wraparound 46 start/stop logging 47 alternate/normal screen buffer 48 reverse/normal status line 49 page/normal scroll mode		 Xv, Xh Xv, Xh Xv, Xh Xv, Xh Xv, Xh Xv, Xh Xv, Xh Xv, Xh Xv, Xh Xv, Xh
tbc	tabulation clear 0 clear horizontal tab stop at active position 1 vertical tab at line indicated by cursor 2 horizontal tabs on line 3 all horizontal tabs 4 all vertical tabs	ESC [Ps g (default Ps = 0)	 Xv, Xh, H, V H H Xv, Xh, H, V H
VTD	virtual terminal data	ESC [x	Xv, Xh, H
VTL	virtual terminal device input	ESC [y	Xh, H
VTR	vt raw keyboard input	ESC [w	Xh, H
vts	vertical tab stop	ESC I	H
xes	erase status line	ESC [? E	Xv, Xh
xrs	return from status line	ESC [? F	Xv, Xh
xhs	hide status line	ESC [? H	Xv, Xh
xss	show status line	ESC [? S	Xv, Xh
xgs	go to column of status line	ESC [? Ps T	Xv, Xh
xst	set text parameters 0 change window name and title to Pt 46 change log file to Pt	ESC] Ps ; Pt \007	Xv, Xh Xv, Xh Xv, Xh

¦ Copyright IBM Corp. 1985, 1991

D.0 Appendix D. Converting X-Windows Calls

Not only do AIX RT X-Windows, Version 2.1, which is X11, and RT X-Windows, Version 1.1, which is X10, offer different function; but, in some cases, both versions of X-Windows offer the same function. Even so, the syntax in these two versions of X-Windows is not always compatible. This appendix, which applies to the RT X-Windows, Version 1.1 only, lists and describes some of these differences.

Note: AIX PS/2 X-Windows licensed program, Version 1.1, is based on X11. It contains the same function as AIX RT X-Windows, Version 2.1.

Subtopics

¦ Copyright IBM Corp. 1985, 1991

D.1 Compatibility Functions

The functions listed here are in **liboldX.a**. Include this library in the compiler command to build your program. For example,

```
  {compiler option} -o samples samples.c -loldX -lX11
```

Subtopics
D.1.1 Drawing Functions
D.1.2 Associating X Resources
D.1.3 Associate Table Functions

¦ Copyright IBM Corp. 1985, 1991

D.1.1 Drawing Functions

Xlib provides functions with which to draw or fill arbitrary polygons or curves. These functions are provided for compatibility with AIX RT X-Windows, Version 2.1, and have no server support. That is, these functions call other **Xlib** routines, instead of the server directly. For example, to draw straight lines, use **XDrawLines** or **XDrawSegments** functions with ease.

Some of the functions discussed provide all the functionality of the RT X-Windows 1.1 routines such as **XDraw**, **XDrawFilled**, **XDrawPatterned**, **XDrawDashed**, and **XDrawTiled**. The new functions are as compatible as possible given the new line drawing routines. Note, however, that **VertexDrawLastPoint**, **XAppendVertex**, and **XClearVertexFlag** are no longer supported. Also, the error status returned by the new functions, which is the AIX RT X-Windows, Version 2.1 standard error status, is the opposite of the error status returned by the old functions.

Also, keep in mind the following:

How the graphics context is set up determines whether or not you ge dashes, and so on.

Lines are properly joined if they connect and include the closing of closed figure. (See **XDrawLines** in Chapter 2, "X-Windows Xlib Functions.")

The functions discussed return a zero under the followin circumstances:

- If they fail because of insufficient memory.

- If a Vertex list with **VertexStartClosed** is passed with a Vertex with **VertexEndClosed** set.

Subtopics
D.1.1.1 XDraw
D.1.1.2 XDrawFilled

¦ Copyright IBM Corp. 1985, 1991

D.1.1.1 XDraw

To achieve the effects of the RT X-Windows 1.1 functions **XDraw**, **XDrawDashed**, and **XDrawPatterned**, use **XDraw**.

```
#include <X11/X10.h>

Status XDraw(display, drawable, gc, vlist, vcount)
        Display *display;
        Drawable drawable;
        GC gc;
        Vertex *vlist;
        int vcount;
```

display Specifies the connection to the X Server.

drawable Specifies the drawable.

gc Specifies the graphics context.

vlist Specifies a pointer to the list of vertices which indicate what to draw.

vcount Specifies the number of vertices in *vlist*.

XDraw draws an arbitrary polygon or curve which is defined by the specified list of Vertexes as specified in *vlist*. The points are connected by lines specified in the flags in the vertex structure.

Each Vertex, as defined in **<X11/X10.h>**, is a structure with the following elements:

```
typedef struct _Vertex {

    short x,y;
    unsigned short flags;

} Vertex;
```

The members of the **Vertex** data structure include the following:

The *x, y* coordinates of the vertex are the following:

Relative to the upper-left inside corner of the drawable i **VertexRelative** is zero.

Relative to the previous vertex if **VertexRelative** is one.

The flags, as defined in **X10.h**, are:

VertexRelative	0x0001	else absolute
VertexDontDraw	0x0002	else draw
VertexCurved	0x0004	else straight
VertexStartClosed	0x0008	else not
VertexEndClosed	0x0010	else not

If **VertexRelative** is not set, the coordinates are absolute (relative to the drawable). The first vertex must be an absolute vertex.

¦ Copyright IBM Corp. 1985, 1991

If **VertexDontDraw** is one, no line or curve is drawn from the previous vertex to this one. This is analogous to picking up the pen and moving to another place before drawing another line.

If **VertexCurved** is one, a spline algorithm is used to draw a smooth curve from the previous vertex, to the next vertex through this vertex. Otherwise, a straight line is drawn from the previous vertex to this one. It is wise to set **VertexCurved** to one only if a previous vertex and the next vertex are both defined either explicitly in the array or through the definition of a closed curve.

VertexDontDraw bits and **VertexCurved** bits can be set to one. This is useful in defining the previous point for the smooth curve and if you do not want an actual curve drawing to start until this point.

If **VertexStartClosed** is one, this point marks the beginning of a closed curve. This vertex must be followed in the array by another vertex whose absolute coordinates are identical with **VertexEndClosed** bit set to one. The points in between form a cycle to determine predecessor and successor vertices for the spline algorithm.

XDraw uses the following graphics context components: *function, plane_mask, line_width, line_style, cap_style, join_style, fill_style, subwindow_mode, clip_x_origin, clip_y_origin*, and *clip_mask*. This function also uses the graphics context mode-dependent components: *foreground, background, tile, stipple, ts_x_origin, ts_y_origin, dash_offset*, and *dash_list*.

¦ Copyright IBM Corp. 1985, 1991

D.1.1.2 XDrawFilled

To achieve the effects of the V10 **XDrawTiled** and **XDrawFilled**, use **XDrawFilled**.

```
#include <X11/X10.h>

Status XDrawFilled(display, drawable, gc, vlist, vcount)
        Display *display;
        Drawable d;
        GC gc;
        Vertex *vlist;
        int vcount;
```

display	Specifies the connection to the X Server.
drawable	Specifies the drawable.
gc	Specifies the graphics context.
vlist	Specifies a pointer to the list of vertices which indicate what to draw.
vcount	Specifies the number of vertices *vlist*.

XDrawFilled draws arbitrary polygons or curves and then fills them. It uses the graphics context components: *function, plane_mask, line_width, line_style, cap_style, join_style, fill_style, subwindow_mode, clip_x_origin, clip_y_origin*, and *clip_mask*. This function also uses the graphics context mode-dependent components: *foreground, background, tile, stipple, ts_x_origin, ts_y_origin, dash_offset, dash_list, fill_style* and *fill_rule*.

¦ Copyright IBM Corp. 1985, 1991

D.1.2 Associating X Resources

It is often necessary to associate arbitrary information with resource IDs. Application programs often need to refer to their own data structures easily when an event arrives. The **XAssocTable** provides you with a method of associating your own data structures with X resources such as bitmaps, pixmaps, fonts, windows, and so on.

An **XAssocTable** can be used to *type* X resources. For example, to create three or four *types* of windows with different properties, each window ID is associated with a pointer to a user-defined *window property* data structure. (A generic type, which is called **XID**, is defined in **Xlib.h**.)

Observe the following guidelines when using an **XAssocTable**:

Be sure the correct display is active before initiating an **XAssocTable** function because all **XID**s are relative to a specified display.

Keep the size of the associate table (number of buckets in the hashin system) to a power of 2 and keep no more than 8 XIDs per bucket to make your associations more efficient.

There is no restriction on the number of XIDs per table, the number of XIDs per display or the number of displays per table.

¦ Copyright IBM Corp. 1985, 1991

D.1.3 Associate Table Functions

Xlib provides the **AssocTable** functions to make such an association. These functions have been replaced by the context management functions.

Subtopics
D.1.3.1 XCreateAssocTable
D.1.3.2 XDeleteAssoc
D.1.3.3 XDestroyAssocTable
D.1.3.4 XLookUpAssoc
D.1.3.5 XMakeAssoc

¦ Copyright IBM Corp. 1985, 1991

D.1.3.1 XCreateAssocTable

```
#include <X11/X10.h>
XAssocTable *XCreateAssocTable(size)
      int size;
```

size Specifies the number of buckets in the hash system of **XAssocTable**.

XCreateAssocTable returns a pointer to a newly created associate table. Use buckets to the power of two to be more efficient, for example, use 32 buckets per 100 objects. (A reasonable maximum number of object per buckets is 8.)

A NULL pointer is returned if there is an error allocating memory for the **XAssocTable**.

¦ Copyright IBM Corp. 1985, 1991

D.1.3.2 XDeleteAssoc

```
#include <X11/X10.h>
XDeleteAssoc(display, table, x_id)
      Display *display;
      XAssocTable *table;
      XID x_id;
```

display Specifies the connection to the X Server.

table Specifies the associate table.

x_id Specifies the XID.

XDeleteAssoc deletes an entry from a specific associate table. It deletes an association in an **XAssocTable** keyed on its XID. Redundant deletes and deletes of non-existent XIDs are meaningless and do not cause problems. Deleting associations does not impair the performance of an **XAssocTable**.

¦ Copyright IBM Corp. 1985, 1991

D.1.3.3 XDestroyAssocTable

```
#include <X11/X10.h>
XDestroyAssocTable(table)
      XAssocTable *table;
```

table Specifies the associate table.

XDestroyAssocTable frees the memory associated with a specific associate table. Using an **XAssocTable** after it has been destroyed will have unpredictable and probably disastrous consequences.

¦ Copyright IBM Corp. 1985, 1991

D.1.3.4 XLookUpAssoc

```
#include <X11/X10.h>
char *XLookUpAssoc(display, table, x_id)
      Display *display;
      XAssocTable *table;
      XID x_id;
```

display Specifies the connection to the X Server.

table Specifies the associate table.

x_id Specifies the XID.

XLookUpAssoc obtains data from a specific assoctable. It retrieves the data stored in an **XAssocTable** by its XID. If an appropriately matching XID is found in the table the routine returns the data associated with it. If the XID cannot be found in the table the routine returns NULL.

¦ Copyright IBM Corp. 1985, 1991

D.1.3.5 XMakeAssoc

```
#include <X11/X10.h>
XMakeAssoc(display, table, x_id, data)
      Display *display;
      XAssocTable *table;
      XID x_id;
      char *data;
```

display Specifies the connection to the X Server.

table Specifies the assocate table.

x_id Specifies the XID.

data Specifies the data to be associated with the *x_id*.

XMakeAssoc creates an entry in a specific assoctable. It inserts data into an **XAssocTable** keyed on an XID. Data is inserted into the table only once. Redundant inserts are meaningless and do not cause problems. The queue in each association bucket is sorted from the lowest XID to the highest XID.

¦ Copyright IBM Corp. 1985, 1991

D.2 RT X-Windows, Version 1.1 and AIX RT X-Windows, Version 2.1 Functions

This section contains a lists of RT X-Windows, Version 1.1 functions with the AIX RT X-Windows, Version 2.1 compatible or equivalent functions. Note the following:

Most AIX RT X-Windows, Version 2.1 functions take a *display* as the first argument, therefore, the display argument is not listed.

Graphics contexts are new in the AIX RT X-Windows, Version 2.1

In RT X-Windows, Version 1.1, **XSetState** sets the foreground, background, function and plane mask in the argument list. In AIX RT X-Windows, Version 2.1, back-to-back graphics calls to the same window using the same GC are merged into poly requests automatically.

Changes to a GC are cached and written only through the server if the GC is used or if the change affects a resource ID based member.

RT X-Windows, Version 1.1	**AIX RT X-Windows, Version 2.1**
DisplayWidth	A macro in **Xlib.h**, which takes *display* and *screen_number* arguments.
DisplayHeight	A macro in **Xlib.h.**, which takes *display* and *screen_number* arguments.
XAddHost	The host argument is **XHostAddress** with family = AF_INET.
XAddNode	**XAddHost** and **XHostAddress** with family = AF_DNET.
XAppendToBuffer	Use **XFetchBuffer** to append this buffer with new string, then call **XStoreBuffer**.
XAppendVertex	Obsolete. See "Drawing Points, Lines, Rectangles, and Arcs" in topic 1.8.3.
XAutoRepeatOn	Same.
XAutoRepeatOff	Same.
XBitmapBitsPut	Use **XPutImage** or, call **XCreateBitmapFromData** followed by **XCopyPlane**.
XCharBitmap	Use **XCreatePixmap**, followed by **XDrawString**. To create a cursor from a font character, use **XCreateGlyphCursor**.
XCharWidths	No direct equivalent. Use **XQueryFont**() followed by **XTextWidth** for each character in the string. Then, store the results in an array of widths.
XCheckMaskEvent	Same function, but different names and definitions for the event masks and event names.

¦ Copyright IBM Corp. 1985, 1991

XCheckWindowEvent	Same function, but different names and definitions for the event masks and event names.
RT X-Windows, Version 1.1	**AIX RT X-Windows, Version 2.1**
XChangeBackground	Use **XSetWindowBackgroundPixmap**. If the background is a solid color, use **XSetWindowBackground**.
XChangeBorder	Use **XSetWindowBorderPixmap**. If the border is a solid color, use **XSetWindowBorder**.
XChangeWindow	Use **XResizeWindow**. To raise the window, use **XConfigureWindow**.
XCircWindowDown	Use **XCirculateSubwindowsDown**.
XCircWindowUp	Use **XCirculateSubwindowsUp**.
XClear	Use **XClearWindow** to clear the entire window, or **XClearArea** to clear part of a window only.
XClipClipped	Windows have a *subwindow mode* as a GC, instead of a *clip mode* attribute. Use **XChangeGC**.
XClipDrawThrough	Use **XSetSubWindowsMode** () to set the GC value of *ClipByChildern* to **XClipClipped** and *IncludeInferior* to **XClipDrawThrough**.
XCloseDisplay	Same.
XCloseFont	Use **XFreeFont**.
XClearIconWindow	Obsolete; replaced by WM_HINTS property argument. Use **XSetStandardProperties** and **XSetWMHints**. See "Setting and Getting Window Manager Hints" in topic 1.12.4.
XCompressEvents	Obsolete. Use **PointerMotionHint** as the event mask. Then, after the **MotionNotify** event, call **XQueryPointer** with *is_hint* set to **True**.
XCondWarpMouse	Use **XWarpPointer**.
XConfigureWindow	Use **XMoveResizeWindow**. To raise the window, use **XConfigureWindow**.
XCopyArea	Use **XCopyArea** to copy between two windows, between a window and a Pixmap, or between two Pixmaps.
XCreate	Obsolete. Use **XCreateSimpleWindow** or **XCreateWindow** and **XSetStandardProperties**.

¦ Copyright IBM Corp. 1985, 1991

XCreateAssoc	Same (**liboldX.a**). Use new Context routines instead. See "Using the Context Manager" in topic 1.14.
XCreateCursor	No equivalent. Use **XCreateGlyphCursor** or **XCreateBitmapFromData**, which makes a single-plane bitmap, or **XCreatePixmapCursor**. See Appendix B, "Xlib Cursor Fonts" in topic B.0.
RT X-Windows, Version 1.1	**AIX RT X-Windows, Version 2.1**
XCreateTerm	Obsolete. Use **XCreateSimpleWindow** or **CreateWindow** and **XSetStandardProperties**.
XCreateTransparency	Does not support transparent windows. Use an **InputOnly** window to define an input region. Use **XSetClipRectangles** to clip output.
XCreateTransparencies	Does not support transparent windows. Use an **InputOnly** window to define an input region. Use **XSetClipRectangles** to clip output.
XCreateWindow	Use **XCreateSimpleWindow**. (Note the background and border are now pixel values rather than Pixmaps.) Or, use **XCreateWindow** which allows the use of nonsolid border, background Pixmaps, initial event masks and cursors, and other variables.
XCreateWindows	Obsolete. Use **XCreateWindow** or **XCreateSimpleWindow** to create windows one at a time.
XCreateWindowBatch	Obsolete. Use **XCreateWindow** or **XCreateSimpleWindow** to create windows one at a time.
XDefineCursor	Same. Or, use **XCreateWindow** and **XChangeWindowAttributes**.
XDeleteAssoc	Same (**liboldX.a**). Use new Context routines. See "Using the Context Manager" in topic 1.14.
XDestAssoc	Same (**liboldX.a**). Use new Context routines. See "Using the Context Manager" in topic 1.14.
XDestroySubwindows	Same.
XDestroyWindow	Same.
XDisplayName	Same.
XDraw	Available in **liboldX.a** library to aid

¦ Copyright IBM Corp. 1985, 1991

	ports, but should be used only to draw a spline. To draw points, lines, rectangles, arcs, or circles, see "Drawing Points, Lines, Rectangles, and Arcs" in topic 1.8.3. The new **XDraw** takes a GC instead of separate arguments for width, function, pixel, planes, etc.
XDrawDashed	Use **XDraw**, after setting the GC's dash list with **XChangeGC** or **XSetDashes**. Or, **XDrawLine(s)**, or **XDrawSegments** or **XDrawRectangle(s)** instead.
RT X-Windows, Version 1.1	**AIX RT X-Windows, Version 2.1**
XDrawFilled	Still supported, but should be used only to fill areas bounded by splines. Otherwise, use **XFillArc(s)**, **XFillPoly**, and **XFillRectangle(s)**.
XDrawPatterned	Use **XDraw**, after setting the GC's dash list with **XChangeGC** or **XSetDashes**. Or, **XDrawLine(s)**, or **XDrawSegments** or **XDrawRectangle(s)** instead.
XDrawTiled	Still supported, but should be used only to fill areas bounded by splines. Otherwise, use **XFillArc(s)**, **XFillPoly**, and **XFillRectangle(s)**.
XErrDescrip	Use **XGetErrorText**.
XErrorHandler	Use **XSetErrorHandler**.
XExpandEvents	Obsolete. To get selected motion events, use **PointerMotionHint** as the event mask. Then, call **XQueryPointer** with *is_hint* set to **True**.
XFeep	Use **XBell**.
XFeepControl	Part of **XChangeKeyboardControl**.
XFetchBuffer	Same.
XFetchBytes	Same.
XFetchName	Same. (Implemented as a window property.)
XFlush	Same.
XFocusKeyboard	Use **XSetInputFocus**.
XFontWidths	Obsolete. Use **XQueryFont** and **XLoadQueryFont**.
XFreeBitmap	**XFreePixmap** (Bitmaps are now single-plane Pixmaps.)

¦ Copyright IBM Corp. 1985, 1991

XFreeColors	Same, but now takes a Colormap argument.
XFreeCursor	Same.
XFreeFont	Use **XUnloadFont**.
XFreePixmap	Same.
XGeometry	Same, but takes *display* and *screen_number* parameters.
XGetColorCells	Use **XAllocColorCells**.
XGetColor	Use **XAllocNamedColor**.
XGetDefault	Same.
XGetFont	Use **XLoadFont**.
XGetHardwareColor	Use **XAllocColor**.
XGetHosts	Use **XListHosts** which returns a list of **XHostAddress** structures.
RT X-Windows, Version 1.1	**AIX RT X-Windows, Version 2.1**
XGetIconWindow	Obsolete. Replaced by WM_HINTS property. See **XGetWMHints**.
XGetNodes	Use **XListHosts** which returns a list of **XHostAddress** structures.
XGetResizeHint	Obsolete. Replaced by WM_SIZE_HINTS properties. See **XGetSizeHints**.
XGrabButton	Same.
XGrabMouse	Use **XGrabPointer**.
XGrabServer	Same.
XIOErrorHandler	Use **XSetIOErrorHandler**.
XInterpretLocator	Use **XTranslateCoords**.
XKeyClickControl	Folded into **XChangeKeyboardControl**.
XLine	Use **XDrawLine** which has a GC argument that replaces width, func, and other graphics components.
XLockToggle	Not supported.
XLockUpDown	Not supported. Use **XSetModifierMapping** to disable the lock key.
XLookupMapping	Same.
XLookUpAssoc	Same (**liboldX.a**). Use new context routines instead. See "Using the

¦ Copyright IBM Corp. 1985, 1991

	Context Manager" in topic 1.14.
XLowerWindow	Same (implemented as a special case of **ConfigureWindow**).
XMakeAssoc	Same (**liboldX.a**). Use new context routines instead. See "Using the Context Manager" in topic 1.14.
XMakePattern	Obsolete. See **XChangeGC** and **XSetDashes**.
XMakePixmap	No equivalent. To make a multi-plane Pixmap from a Bitmap, use **XCopyPlane** to transfer the Bitmap (now a single-plane Pixmap) directly to the destination window. To make a multi-plane Pixmap from a single-plane Bitmap, use **XCreatePixmap** followed by **XCopyPlane**.
XMakeTile(s)	No equivalent. To make a solid border or background tile, set the border or background directly to the pixel value without making a tile Pixmap first. To set a solid color Pixmap, use **XCreatePixmap** followed by **XFillRectangle**. (See **XPixFill**)
XMapSubwindows	Same.
RT X-Windows, Version 1.1	**AIX RT X-Windows, Version 2.1**
XMapWindow	Same, but does not raise the windows. Use **XMapRaised** to map and raise a window.
XMaskEvent	Same, but different names and definitions for the event masks and event names.
XMouseControl	Use **XChangePointerControl**.
XMoveArea	Use **XCopyArea**.
XMoveWindow	Same, but does not raise windows. Use **XConfigureWindow** to raise a window.
XNextEvent	Same, but different names and definitions for the event masks and event names.
XOpenDisplay	Same, but string argument may include a screen number after a decimal point, for example, **TRUC:1.0**. If the screen number is omitted, it defaults to zero.
XOpenFont	Use **XLoadQueryFont**.
XParseColor	Same.

¦ Copyright IBM Corp. 1985, 1991

XParseGeometry	Same.
XPeekEvent	Same, but string argument may include a screen number after a decimal point, for example, **TRUC:1.0**. If the screen number is omitted, it defaults to zero.
XPending	Same.
XPixmapBitsPutXY	Use **XPutImage** with *format* set to XYPixmap.
XPixmapBitsPutZ	Use **XPutImage** with *format* set to ZPixmap.
XPixFill, XPixSet	Use **XFillRectangle** after setting the foreground pixel and fill-style of the GC with **XChangeGC**, or the **XSetForeground** and **XSetFillStyle**. Set the fill-style to **FillSolid**.
XPixmapGetXY	Use **XGetImage** with *format* set to XYPixmap.
XPixmapGetZ	Use **XGetImage** with *format* set to ZPixmap.
XPixmapPut	Use **XCopyArea**.
XPixmapSave	Obsolete.
XPutBackEvent	Same.
XQueryBrushShape	Obsolete, brushed lines are no longer supported.
XQueryColor(s)	Same, but takes a Colormap argument.
XQueryCursorShape	Use **XQueryBestCursor**, which takes a *drawable* as an additional argument to identify a screen for the query.
RT X-Windows, Version 1.1	**AIX RT X-Windows, Version 2.1**
XQueryFont	Similar, but it returns an **XFontStruct** structure, which includes the width of the array for a variable-width font.
XQueryInput	Use **XGetWindowAttributes**.
XQueryMouse	Use **XQueryPointer**.
XQueryMouseButtons	Use **XQueryPointer**.
XQueryTileShape	Use **XQueryBestTile**, which takes a *drawable* to identify a screen and depth for the query.
XQueryTree	Same, but now also returns the root

¦ Copyright IBM Corp. 1985, 1991

	window of the screen that contains the window.
XQueryWidth	Use **XQueryTextExtents** which returns an **XFontStruct** data structure. Use **XQueryFont** or **XLoadQueryFont** to get the **XFontStruct** data structure before using **XTextExtents** or **XTextWidth**. Use **XTextExtents** or **XTextWidth** if the **XFontStruct** data structure is available. These functions take the **XFontStruct *** structure as an argument and compute the dimensions on the client side.
XQueryWindow	Use **XGetGeometry** or **XGetWindowAttributes** and **XGetWMHints** depending on the information to be returned.
XRaiseWindow	Same, but implemented as a special case of **ConfigureWindow**.
XReadBitmapFile	Same, but takes a *drawable* and returns a Pixmap of depth 1 on the same screen as the drawable. The AIX RT X-Windows, Version 2.1 bitmap file format is different from the bitmap file format of RT X-Windows Version 1.1, but this function reads both format.
XRebindCode	Same.
XRemoveHost	Same.
XRemoveNode	Use **XRemoveHost**.
XRotateBuffers	Same.
XScreenSaver	Use **XSetScreenSaver**.
XSelectInput	Same, or, use **XCreateWindow** and **XChangeWindowAttributes**.
XSetDisplay	Obsolete, there is no global current display.
XSetIconWindow	Obsolete, replaced by WM_HINTS property. See **XSetStandardProperties** and **XSetWMHints**.
RT X-Windows, Version 1.1	**AIX RT X-Windows, Version 2.1**
XSetResizeHint	Obsolete, replaced by WM_SIZE_HINTS properties. See **XSetStandardProperties** and **XSetSizeHints**.
XStippleFill	Use **XFillRectangle**, but first set the foreground pixel, the stipple pattern,

¦ Copyright IBM Corp. 1985, 1991

	and fill-style in the GC with **XChangeGC** or **XSetForeground**, **XSetStipple**, and **XSetFillStyle**. Set the fill-style to **FillStippled**.
XStoreBitmap	Use **XCreateBitmapFromData**. The order of the arguments and the format of the data is different. In the present form, **char *** is an 8-bit value rather than 16-bit padding. Also, it takes a *drawable* as an argument to identify which screen owns the bitmap. Use the **bm-convert** example program to convert bitmap data from 16-bit to 8-bit format. If you must use 16-bit format data, call **XCreatePixmap** (with depth == 1) followed by **XPutImage** (with image->depth == 1, image->format == XYBitmap, image->byte_order == image->bitmap_bit_order == LSBFirst, image->bitmap_unit == image->bitmap_pad == 16).
XStoreBuffer	Same.
XStoreBytes	Same.
XStoreColor(s)	Same.
XStoreCursor	Use **XCreatePixmapCursor**.
XStoreName	Same.
XStorePixmapXY	Use **XCreatePixmap** followed by **XPutImage** with *format* set to XYPixmap.
XStorePixmapZ	Use **XCreatePixmap** followed by **XPutImage** with *format* set to ZPixmap.
XStringWidth	Use **XTextWidth** which takes **XFontStruct *** and a string, and computes the width without querying the server. Character and space padding are no longer supported. The string argument is not null-terminated; instead, supply a byte-count.
XSync	Same.
RT X-Windows, Version 1.1	**AIX RT X-Windows, Version 2.1**
XText, XTextPad	Use **XDrawImageString**, but first set the foreground, background, and font in the GC with **XChangeGC** or **XSetForeground**, **XSetBackground**, and **XSetFont**. Character and space padding is no longer supported.

¦ Copyright IBM Corp. 1985, 1991

	The *x* and *y* arguments refer to the ORIGIN of the first character, NOT to the top-left corner. The *y* component of the origin is the character's baseline. The *ascent* field in the **XFontStruct** structure gives the distance from the top of the font to the baseline.
XTextMask, XTextMaskPad	Use **XDrawString**, but first set the foreground, function, font, fill-style, and other attributes of the GC as appropriate. Character and space padding is not supported in the protocol. Use **XDrawText16** to draw multiple strings, possibly in multiple fonts, with arbitrary spacing between the strings. The *x* and *y* arguments refer to the ORIGIN of the first character, NOT to the top-left corner. The *y* component of the origin is the character's baseline. The *ascent* field in the **XFontStruct** structure gives the distance from the top of the font to the baseline.
XTileAbsolute, XTileRelative	No longer supported. A window's background tile origin is aligned with the window origin, unless the background is **ParentRelative**, in which case the origin of the background tile is aligned with the origin of the background tile of the parent. The border tile origin of a window is the same as the background tile origin. When you use **XFillArc(s)**, **XFillPolygon**, or **XFillRectangle(s)** to fill an area, the tile origin is determined by the *ts-x-origin* and *ts-y-origin* attributes of the GC.
XTileFill, XTileSet	Use **XFillRectangle**, but first set the foreground and background pixels and fill-style of the GC with **XChangeGC**, or **XSetForeground**, **XSetBackground**, **XSetTile**, and **XSetFillStyle**. Fill-style should be set to **FillTiled**. If you are using a clip-mask, change this attribute with **XChangeGC** or **XSetClipMask**.
XUndefineCursor	Same, or use **XChangeWindowAttributes** to set the cursor to None.
XUngrabButton	Same, but requires a Window argument.

¦ Copyright IBM Corp. 1985, 1991

RT X-Windows, Version 1.1	AIX RT X-Windows, Version 2.1
XUngrabMouse	Use **XUngrabPointer**.
XUngrabServer	Same.
XUnmapSubwindows	Same.
XUnmapTransparent	No longer supported.
XUnmapWindow	Same.
XUpdateMouse	Obsolete. Use **PointerMotionHint** as the event mask. After the **MotionNotify** event, call **XQueryPointer** with *is_hint* set to **True**.
XUseKeymap	Same.
XWarpMouse	Use **XWarpPointer** with the following values: source window set to None and src_x, src_y, src_width, and src_height set to zero.
XWindowEvent	Same, but different names and definitions for the event masks and event names.

¦ Copyright IBM Corp. 1985, 1991

E.0 Appendix E. Writing a Client

To enable window managers to handle the competing demands of resources, such as screen space, both clients and window managers must adhere to certain conventions.

In general, these conventions are complex. Undoubtedly, they change as new window management paradigms are developed. Therefore, only conventions that are essential and general to all window management paradigms are defined. Clients designed to run with a particular window manager can easily define additional, private protocols, but must be aware that users might choose to run another window manager.

A principle of these conventions is that a general client should neither know nor care which window manager is running, if any. As part of this principle, each window manager implements a window management policy chosen by the user or the system administrator. The choice of an appropriate policy is not made for an individual client, but clients can use private protocols to restrict themselves to operating only under a specific window manager. This policy merely ensures that no claim of general utility is made for such programs.

Window management policy should provide flexibility in handling clients. A client is mismanaged if it is given, for example, unqualified preferential treatment because of its presumed importance. This kind of policy is too inflexible when a client requires a lower priority, for example when the same client is being debugged. Instead, the client should be run under a window manager that allows other windows, such as the debugger, to appear on top when appropriate. (1)

A client that requires information about the resources of a window manager, the *res_class* value of the window manager is **Wm**. The *res_name* is the name of the window manager (for example, **aixwm**). This differs from the normal convention, for two reasons:

Clients should be able to discover the resources that a window manage is using without being forced to know which window manager it is.

At most one (top-level) window manager should be running, even if th server is driving several screens.

(1) Permission to use, copy, modify and distribute this document for any purpose and without fee is hereby granted, provided that the above copyright notice and this permission notice appear in all copies, and that the name of Sun Microsystems, Inc. not be used in advertising or publicity pertaining to distribution of the software without specific, written prior permission. Sun Microsystems, Inc. makes no representations about the suitability of the software described herein for any purpose. It is provided "as is" without express or implied warranty.

Subtopics

¦ Copyright IBM Corp. 1985, 1991

¦ Copyright IBM Corp. 1985, 1991

E.1 Defining Client Actions

In general, the object of the X-Windows design is for clients to act the same way with or without the window manager. To accomplish this, clients should do the following:

Hint to the window manager about which resources they want to obtain

Cooperate with the window manager by accepting the resources they ar allocated, even if these resources were not requested.

Be prepared for resource allocations to change at any time

¦ Copyright IBM Corp. 1985, 1991

E.2 Creating a Top-Level Window

A client normally expects to create its top-level windows as children of one or more of the root windows, for example:

```
window = XCreateSimpleWindow(display, DefaultRootWindow(display),
                  xsh.x, xsh.y, xsh.width, xsh.height, bw, bd, bg);
```

or, a particular root window:

```
window = XCreateSimpleWindow(display, RootWindow(display, screen),
                xsh.x, xsh.y, xsh.width, xsh.height, bw, bd, bg);
```

Ideally, it should be possible to override the choice of a root window and allow clients, including window managers, to treat a non-root window as a pseudo-root. This would allow, for example, testing of window managers and the use of application specific window managers to control the sub-windows owned by the members of a related suite of clients.

To support this, the following extension to the semantics of **XOpenDisplay()** on AIX-based systems is defined. (Similar extensions are required for other operating systems.) On AIX-based systems, the display name or **DISPLAY** environment variable is a string that has the format:

hostname:number.screen

¦ Copyright IBM Corp. 1985, 1991

E.3 Defining Client Properties

Once the client has one or more top-level windows, it must place properties on that window to inform the window manager of the desired behavior. Some of these properties are mandatory and some are optional. Properties written by the client should not be changed by the window manager.

Client top-level window properties include **WM_CLASS**, **WM_HINTS**, **WM_ICON_NAME**, **WM_NAMES**, **WM_NORMAL_HINTS**, and **WM_TRANSIENT_FOR**.

Subtopics

¦ Copyright IBM Corp. 1985, 1991

E.3.1 WM_CLASS

The **WM_CLASS** property is a string that contains two null-separated elements, *res_class* and *res_name*, used by clients to identify and obtain resources related to the window.

The *res_class* element identifies the clients. It is set to the name of the client, for example, **emacs**.

The *res_name* element identifies the particular instance more specifically. It is set to one of the following:

- An optional command line argument (*-m* name)
- A specific environment variable (*RESOURCE_NAME*)
- A trailing component of *argv[0]*.

The *res_name* field is set to the first option that applies. Use *res_name* first to obtain resources. If it is not successful, then use *res_class*.

The **WM_CLASS** property is write-once and must be present when the window is mapped. Window managers ignore changes to it while the window is mapped. However, there should be no reason for a client to change its class.

¦ Copyright IBM Corp. 1985, 1991

E.3.2 WM_HINTS

The **WM_HINTS** property communicates required information to the window manager. This property does not communicate the following information:

Window geometry, which is available from the windo
The constraints of the geometry, which is available from th **WM_NORMAL_HINTS** structure
Various strings, which need separate properties such as **WM_NAME**.

The **XWMHints** data structure is defined as:

```
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;
```

A mapped window without a **WM_HINTS** property is in the **Normal** state. An unmapped window without a **WM_HINTS** property is in the **Ignored** state.

The members of the **XWMHints** structure include:

The *flags* field which is defined as:

```
#define    InputHint            (1L << 0)
#define    StateHint            (1L << 1)
#define    IconPixmapHint       (1L << 2)
#define    IconWindowHint       (1L << 3)
#define    IconPositionHint     (1L << 4)
#define    IconMaskHint         (1L << 5)
#define    WindowGroupHint      (1L << 6)
```

The *input* field which communicates the type of input focus used by the client to the window manager. The different types of input focus are:

No Input

The client never expects keyboard input, for example **xload**, or another output-only client. Clients with **No Input** should set *input* to **FALSE**.

Passive Input

The client expects keyboard input, but never sets the input focus explicitly. One example is a client without subwindows that accepts input in **PointerRoot** mode. Another example is when the window manager sets the input focus to its top-level window.

Clients with **Passive Input** should set *input* to **TRUE**.

Locally Active Input

The client expects keyboard input, and sets the input focus explicitly

¦ Copyright IBM Corp. 1985, 1991

when one of its windows already has the focus. One example is a client with subwindows defining various data entry fields that uses the tab key to move the input focus between the fields after its top-level window has acquired the focus in **PointerRoot mode** or when the window manager sets the input focus to its top-level window.

Clients with **Locally Active** should set *input* to **TRUE**.

Globally Active Input

The client expects keyboard input and sets the input focus explicitly, even when it is in a window not owned by the client. An example is a client with a scrollbar that wants to allow users to scroll the window without disturbing the input focus even when the client is in another window. The client wants to acquire the input focus when the user clicks in the scrolled region, but not when the user clicks in the scrollbar. Thus, it wants to prevent the window manager from setting the input focus to any of its windows.

Clients with **Globally Active** should set *input* to **FALSE**.

The definitions for the *initial_state* flag are:

```
#define    DontCareState     0
#define    NormalState       1
#define    ClientIconState   2
#define    IconicState       3
#define    IgnoreState       5
```

The *initial_state* field which determines the state the client wants to be in the next time the top-level window is mapped. The different states are:

DontCareState

The client does not care what state it is in.

NormalState

The client top-level is in a visible state.

ClientIconState

The clients wants its *icon_window* to be visible. If icon_windows are not available, then, it wants its top-level window visible.

IconicState

The client is in an iconic state. It can assume that its *icon_window*, or *icon_pixmap*, or **WM_ICON_NAME** will be visible.

IgnoreState

The client is invisible and does not need an icon or menu entry to be provided by the window manager.

A top-level window that is mapped or unmapped with **IgnoreState** as the *initial_state* in the **WM_HINTS** property will be treated as if it did not have a **WM_HINTS** property. If it is mapped, it will be put in a **Normal** state. If it is unmapped, it will be put in an **Ignore** state.

¦ Copyright IBM Corp. 1985, 1991

The *icon_pixmap* field which specifies a pixmap as an icon if the pixmap is:

One of the sizes specified in the **WM_ICON_SIZES** property on the root or pseudo-root

One bit deep; the window manager selects, through the default database, a suitable background (for the 0 bits) and foreground (for the 1 bits) colors. These defaults can specify different colors for the icons of different clients.

The *icon_mask* field which specifies the pixels of the *icon_pixmap* to be used as the icon, allowing for non-rectangular icons.

The *icon_window* field which is the ID of the window that the client selects as an icon. Most window managers support icon windows. (Window managers that do not support icon window usually have a user interface in which small windows behave like icons, but are completely inappropriate. Clients should not try to remedy the omission by working around it.)

Window managers that support icon windows expect that:

The icon window is a child of the root or pseudo-root

The icon window is one of the sizes specified in the **WM_ICON_SIZES** property on the root or pseudo-root.

The icon window uses the root visual and default colormap for th screen in question.

The client will not unmap the icon window. Clients that unmap thei icon windows can have irritating visual behavior.

Clients that need more capabilities from the icons than a simple two-color bitmap should use icon windows.

The *window_group* field which lets the client declare the window as a member of a group of windows. Thus, a single client can manipulate multiple children of the root window.

Set the *window_group* field to the ID of the group leader. The window group leader can be a window that exists for that purpose only. A window group leader that is also a place-holder group leader would never be mapped by either the client or the window manager. The properties of the window group leader are the properties for the group as a whole (for example, the icon to be shown when the entire group is iconified) is the icon of the group leader.

Window managers can provide facilities for manipulating the group as a whole. Clients can manipulate the group as a whole by manipulating the leader. Mapping or unmapping the leader causes all the group members to undergo similar transitions. State transitions on the group leader cause corresponding state transitions on all group members. Mapping or unmapping a group member other than the leader (a group follower) affects only the state of the follower.

¦ Copyright IBM Corp. 1985, 1991

E.3.3 WM_ICON_NAME

The **WM_ICON_NAME** property is an uninterpreted string that the client displays in association with the window when it is iconified, for example an icon label. In other respects, it is similar to **WM_NAME**.

¦ Copyright IBM Corp. 1985, 1991

E.3.4 WM_NAME

The **WM_NAME** property is an uninterpreted string that the client displays in association with the window, for example a window titlebar.

The encoding for this string, and all uninterpreted string properties, is implied by the type of the property. The **STRING** type implies ISO Latin-1 encoding.

Window managers are expected to display this information. Clients can assume that at least the first part of this string is visible to the user, and that, if the information is not visible to the user, it is because the user has decided to make it invisible.

On the other hand, there is no guarantee that the user can see the **WM_NAME** string even if the window manager supports window titlebars. The user could have placed the titlebar off-screen or have covered it by other windows. **WM_NAME** should not be used for application-critical information nor to announce asynchronous changes of application state that require timely user response. The expected uses are:

Permit the user to identify one of a number of instances of the sam client.

Provide the user with non-critical state information

Even window managers that support titlebars have a limit on the length of the string that is visible.

¦ Copyright IBM Corp. 1985, 1991

E.3.5 WM_NORMAL_HINTS

The **WM_NORMAL_HINTS** property is an **XSizeHints** structure describing the desired window geometry.

```
typedef struct {

  long flags;
  int x, y
  int width, height
  intmin_width, min_height;
  int max_width, max_height;
  int width_inc, height_inc;
  struct {
    int x;                              /* numerator          */
    int y;                              /* denominator        */
  } min_aspect, max_aspect;
  int base_width, base_height;
} XSizeHints;
```

The *flags* field are defined:

```
#define             (1L << 0)     /* user-specified x, y              */
USPosition
#define USSize      (1L << 1)     /* user-specified width, height     */
#define PPosition   (1L << 2)     /* program-specified position       */
#define PSize       (1L << 3)     /* program-specified size           */
#define PMinSize    (1L << 4)     /* program-specified minimum size   */
#define PMaxSize    (1L << 5)     /* program-specified maximum size   */
#define             (1L << 6)     /* program-specified resize         */
PResizeInc                        increments
#define PAspect     (1L << 7)     /* program-specified min and max    */
                                  aspect ratios
#define PBaseSize   (1L << 8)     /* program-specified base size      */
```

To indicate that the size and position of the window when first mapped were specified by the user, set the **USPosition** and **USSize** flags.

To indicate that the size and position were specified by the client without user involvement, set **PPosition** and **PSize**. This allows a window manager to know that the user specifically asked where the window should be placed or how the window should be sized and that the window manager does not have to rely on the program.

The *x, y, width,* and *height* describe the desired position and size for the window. The *x, y* coordinates indicate the pseudo-root window, regardless of any reparenting that may have occurred.

The *min_width* and *min_height* elements specify the minimum size that the window can be for the client to be useful. The *max_width* and *max_height* elements specify the maximum size. The *base_width* and *base_height* elements in conjunction with *width_inc* and *height_inc* define an arithmetic progression of preferred window widths and heights:

```
  width  = base_width  + ( i * width_inc )
  height = base_height + ( j * height_inc )
```

for non-negative integers *i* and *j*. Window managers are encouraged to use

¦ Copyright IBM Corp. 1985, 1991

i and *j* instead of *width* and *height* in reporting window sizes to users. If a base size is not provided, the minimum size is to be used in its place, and vice versa.

The *min_aspect* and *max_aspect* elements which are expressed as ratios of *x* and *y*, allow a client to specify the range of aspect ratios it prefers.

¦ Copyright IBM Corp. 1985, 1991

E.3.6 WM_TRANSIENT_FOR

The **WM_TRANSIENT_FOR** property is the ID of another top-level window. The implication is that this window is a pop-up on behalf of the named window and window managers can decide not to decorate transient windows or in other ways treat them differently. Dialog boxes are an example of windows that should have **WM_TRANSIENT_FOR** set.

Do not confuse **WM_TRANSIENT_FOR** with override-redirect. **WM_TRANSIENT_FOR** should be used when the pointer is not grabbed while the window is grabbed. In other words, use it when windows are allowed to be active while the transient is up. If other windows must be prevented from processing input, use override-redirect and grab the pointer while the window is mapped. Pop-up menus, for example, normally work this way.

¦ Copyright IBM Corp. 1985, 1991

E.4 Window Manager Properties

This section describes the property that the *window manager* places on client top-level windows.

A window manager property is **WM_STATE**.

Subtopics
E.4.1 WM_STATE

¦ Copyright IBM Corp. 1985, 1991

E.4.1 WM_STATE

This property communicates between window and session managers and should not require clients to examine this property.

The **WM_STATE** property is composed of two fields, *state* and *icon*.

```
struct {
      int state;
      Window icon
};
```

The *state* field can take on some of the same values as the *initial_state* field of the **WM_HINTS** property. In particular, it can be:

```
#define NormalState          1
#define ClientIconState      2
#define IconicState          3
#define IgnoreState          5
```

The *icon* field should contain the window ID for the icon window used by the window manager for the window on which this **WM_STATE** is set, if any. Otherwise the value is **None**. Note that this window may not be the same as the icon window which the client specifies.

The *state* field reflects the state of the window as determined by the window manager, which may not match the state as determined by the client and expressed in the *initial_state* field of the **WM_HINTS** property. If *state* is

NormalState, the window manager expects the client to animate its window.

IconicState, window manager expects the client to animate its icon window.

In either state, clients should be prepared to handle exposure events from either window.

¦ Copyright IBM Corp. 1985, 1991

E.5 Mapping and Unmapping the Window

Once the top-level window has with suitable properties, the client is free to map and unmap as required. Mapping a window places it in **Normal** state. Unmapping a window places it in **Iconic** state unless:

the window has no **WM_HINTS** property, or

the *initial_state* field in the **WM_HINTS** property is **IgnoreState**.

In both cases, the window is placed in **Ignore** state, and the window manager does not display an icon or any other way of re-mapping the window.

The client selects **StructureNotify** on the top-level window and receives an **UnmapNotify** event when it moves to **Iconic** state. The client selects a **MapNotify** when it moves to **Normal** state. This implies that a reparenting window manager will unmap the top-level window as well as the parent window when going **Iconic**.

If the window is a group follower, in addition to unmapping the window itself, a synthetic **UnmapNotify** event must be sent to its parent using **SendEvent** with the following parameters:

destination: the parent of the window
propogate: **FALSE**
event-mask: **SubstructureRedirect** or **SubstructureNotify**
event: an **UnmapNotify** with:
event: the parent of the window
window: the window
from-configure: **FALSE**

Sending this event insures that unmapping a group follower causes a state transition on that window, even though it could have been unmapped as a result of a group-iconify caused by unmapping the leader.

A client can also select **VisibilityChange** on the top-level or icon windows. The windows then receive a **VisibilityNotify**(*state*=**FullyObscured**) event when the subject window becomes completely obscured even though mapped (and thus perhaps a waste of time to update), and a **VisibilityNotify**(*state!*=**FullyObscured**) when it becomes viewable.

Clients should not unmap their icon windows.

¦ Copyright IBM Corp. 1985, 1991

E.6 Configuring the Window

Clients can resize and reposition their top-level windows using the following method:

1. Use the **ConfigureWindow** request (**XConfigureWindow()** in **Xlib**).

2. Then, use the **SendEvent** request (**XSendEvent()** in **Xlib**) to send a synthetic **ConfigureRequest** event. The **SendEvent** request should have the following parameters:

destination:	the parent of the window
propogate:	**FALSE**
event-mask:	**SubstructureRedirect** or **SubstructureNotify**
event:	a **ConfigureRequest** with: *event*: the parent of the window *window*: the window *x,y*: taken **ConfigureWindow** request *width, height*: Same as *x, y* *border-width*: Same as *x, y* *sibling*: Same as *x, y* *stack-mode*: Same as *x, y* *value-mask*: Same as *x, y*

The order is important. The coordinate system in which the location is expressed is that of the root or pseudo-root, regardless of any reparenting that may have occurred.

Clients must be prepared for *any* size and location since there is no guarantee that the window manager will allocate them the requested size or location. If the window manager decides to respond to a **ConfigureRequest** by:

Not changing the size or location of the window at all, a clien which has requested notification by setting the **ConfigureDenied** bit in **WM_HINTS** receives a synthetic **ConfigureNotify** event describing the (unchanged) state of the window. The *x, y* coordinates will be in the root or pseudo-root system, regardless of any reparenting that has taken place. The coordinates will not receive a real **ConfigureNotify**, event since no change has actually taken place.

Moving the window without resizing it, a client which has requeste notification by setting the **WindowMoved** bit in **WM_HINTS** will receive a synthetic **ConfigureNotify** event describing the new state of the window, whose *x,y* coordinates will be in the root or pseudo-root coordinate system. The coordinates cannot receive a real **ConfigureNotify** event describing this change, since the window manager can have reparented their window.

Moving and resizing the window, a client which has selected fo **StructureNotify** receives a **ConfigureNotify** event. Note that the coordinates in this event are relative to the parent, which may not be the root in the window has been reparented.

As a rule, coordinates in real **ConfigureNotify** events are in the parent space, whereas in synthetic events they are in the root space.

¦ Copyright IBM Corp. 1985, 1991

E.7 Setting Input Focus

Clients can, as described above, deal with the input focus in four ways:

No Input
Passive
Locally Active
Globally Active.

Passive and **Locally Active** clients set the *input* field of **WM_HINTS** to indicate that they require window manager assistance in acquiring the input focus. **No Input** and **Globally Active** clients set the *input* field to prevent the window manager setting the input focus to its top-level window.

Clients using **SetInputFocus** must set the *time* field to the timestamp of the event that caused them to make the attempt.

Note: This cannot be a **FocusIn** event, which does not have timestamps. Clients can acquire the focus without a corresponding **EnterNotify**. Clients must not use **CurrentTime** in the *time* field.

Clients using the **Globally Active** model can only use **SetInputFocus** to acquire the input focus when they do not already have it on receipt of one of the following events:

ButtonPress

ButtonRelease

Passive-grabbed **KeyPress**

Passive-grabbed **KeyRelease**.

In general, clients should avoid using passive-grabbed **Key** events for this purpose except when they are unavoidable (as, for example, a selection tool that establishes a passive grab on the keys that cut, copy, or paste).

The method by which the user commands the window manager to set the focus to a window is up to the window manager. For example, clients cannot determine whether they will see the click that transfers the focus.

Clients which set the **FocusMessage** bit in **WM_HINTS** might receive a **ClientMessage** from the window manager whose *type* field is the atom **WM_TAKE_FOCUS** and whose data field is a timestamp. If they expect input, clients set this way should respond with a **SetInputFocus** request with the

window field set to the previous window with input focus or set to the *default input window*

the *time* field set to the timestamp.

Clients normally receive **WM_TAKE_FOCUS** when they open from an icon or when the user clicks outside the window where the window manager assigns the focus (for example, clicking in the title bar bar can be used to assign the focus).

The goal is to support window managers that want to assign the input focus to a top-level window in such a way that the top-level window can either

¦ Copyright IBM Corp. 1985, 1991

assign it to one of its subwindows or decline the offer of the focus. For example, a clock or a text editor without frames currently opened might not take the focus even though the window manager wants the clients to take the input focus after being de-iconified or raised.

The **WM_TAKE_FOCUS** property is not required if the **FocusIn** event contains a timestamp and a *previous_focus* field.

Clients that set the input focus must have a value for the *revert_to* field of the **SetInputFocus** request. This determines the behavior of the input focus if the window to which the focus has been set becomes not viewable. The value can be any of the following:

Parent

In general, clients should use this value when assigning focus to one of their subwindows. Unmapping the subwindow causes focus to revert to the parent.

PointerRoot

Using this value with a **ClickToType**-style window manager may lead to race conditions, since the window becoming unviewable can coincide with the window manager deciding to move the focus elsewhere.

None

Using this value causes problems if the window manager reparents the window (most window managers will) and then crashes. The input focus is **None** and is not likely to be changeable.

A tentative convention for setting client input focus is: Clients invoking **SetInputFocus** should set *revert_to* to **Parent**.

A convention is also required for clients that want to give up the input focus. A tentative convention is: A client that wants to remove the focus should set *revert_to* to **None**, rather than to **PointerRoot**.

Clients can be notified about input focus. By selecting **FocusChange** on their top-level windows, **FocusIn** and **FocusOut** events are received. Clients should not set the input focus to a subwindow unless the focus is set in one of their top-level windows. Clients should set the focus and leave the pointer alone and not warp the pointer in an attempt to transfer the focus.

Once a client has the focus in one of its windows, it can transfer it to another of its windows using:

SetInputFocus

focus: **WINDOW** or **PointerRoot** or **None**

revert_to: {**Parent, PointerRoot, None**}

time: **TIMESTAMP** or **CurrentTime**

Clients using **SetInputFocus** must set the *time* field to the timestamp of the event that caused them to make the attempt.

Note: A **FocusIn** event does not have timestamps. Clients can acquire the

¦ Copyright IBM Corp. 1985, 1991

focus without a corresponding **EnterNotify**. Clients must not use **CurrentTime** in the *time* field.

¦ Copyright IBM Corp. 1985, 1991

E.8 Installing Colormaps

Window managers are responsible for ensuring that top-level windows colormaps are installed at appropriate times. To change the colormap, the client should change the window attribute that relies on the window manager to install the colormap. Set the ID in the *colormap* field of the window attributes. Do not set a **WM_COLORMAPS** property on the top-level window.

However, a client with different colormap requirements for the subwindows and the top-level windows is responsible for installing colormaps on the subwindows. The window manager is responsible for installing the top-level colormap. By setting the **WM_COLORMAPS** property on the top-level window to a list of the colormap IDs to be installed, the window manager is informed, the client installs its own subwindows colormap, and the client allows the window manager to uninstall other maps at suitable times.

Clients, especially those installing their own colormaps, should be aware of the *min-installed-maps* and *max-installed-maps* fields of the connection startup information, and the effect that the minimum value has on the *required list*:

If a subset of the installed maps is an ordered list (the required list), the length of the required list is at most *M*, where *M* is the *min-installed-maps* for the screen in the connection setup. The required list is maintained as follows:

> When a colormap is an explicit argument to **InstallColormap**, it is added to the head of the list. The list is truncated at the tail to keep the length of the list to *M*.
>
> When a colormap is an explicit argument to **UninstallColormap** and it is in the required list, it is removed from the list. A colormap is not added implicitly by the server to the required list when it is installed. Nor is a colormap removed from the required list implicitly by the server when it is uninstalled.
>
> The value of *min-installed-maps* is the number of maps guaranteed to be installed. This number is often one.

The **WM_COLORMAPS** property is merely a hint to the window manager, allowing it to uninstall suitable maps when preparing to install the map of a top-level window. If it is inconvenient for a client to collect the complete set of colormaps to be installed, the property can be incomplete (or even empty). The only result of an incomplete list is that a window manager's attempts to manage the set of installed maps will in some cases be less than optimal.

¦ Copyright IBM Corp. 1985, 1991

E.9 Requesting Icons

A client can hint to the window manager about the desired appearance of its icon in several ways:

Set a string in **WM_ICON_NAME**. All clients should do this to provide a fall-back for window managers whose ideas about icons differ widely from those of the client.

Set a pixmap into the *icon_pixmap* field of the **WM_HINTS** property, and possibly another into the *icon_mask* field. The window manager is expected to display the pixmap masked by the mask. The pixmap should be one of the sizes found in the **WM_ICON_SIZE** property on the root or pseudo-root. If this property is not found, the window manager is unlikely to display icon pixmaps. Window managers will normally clip or tile pixmaps which do not match **WM_ICON_SIZE**.

Set a window into the *icon_window* field of the **WM_HINTS** property. The window manager is expected to map that window whenever the client is in **IconState** or **ClientIconState**. In general, the size of the icon window should be one of those specified in **WM_ICON_SIZE** on the root or pseudo-root, if it exists. Window managers are free to resize icon windows.

Clients can ask the window manager to change their state from **Normal** to **Iconic** and vice versa by mapping and unmapping their top-level window.

Clients must not depend on being able to receive input events via their icon windows. Window managers will differ as to whether they support this; most allow some subset of the keys and buttons through.

¦ Copyright IBM Corp. 1985, 1991

E.10 Creating Pop-up Windows

To pop up a window, clients take one of three actions:

Create and map another normal top-level window to be decorated an managed normally by the window manager. See the following discussion on window groups.

Set the **WM_TRANSIENT_FOR** property. Use this if the window will be visible for a relatively short time and deserves a somewhat lighter treatment. Expect all the normal window manager properties on the window except for less decoration. A dialog box is an example.

Set *override-redirect* on the window. Use this if the window will be visible for a very short time and should not be decorated at all. Generally, this is used only if the pointer is grabbed while the window is mapped, for example, a pop-up menu. The window manager will never interfere with these windows.

¦ Copyright IBM Corp. 1985, 1991

E.11 Defining Window Groups

A client with multiple, persistent top-level windows constitutes a window group. Its top-level windows should be linked together using the *window_group* field of the **WM_HINTS** structure.

The window to which the others point is the group leader. This window carries the group properties, not the individual properties. Window managers can treat the group leader differently from other windows in the group. For example, group leaders can have the full set of decorations while other group members might have a restricted set.

The client does have to map the group leader since it could be a window that exists solely as a place-holder.

Subtopics
E.11.1 Responding to Window Manager Actions

¦ Copyright IBM Corp. 1985, 1991

E.11.1 Responding to Window Manager Actions

The window manager performs a number of operations on client resources, primarily on their top-level windows. Clients should not try to change this, but should try to receive notification of the window manager's operations.

¦ Copyright IBM Corp. 1985, 1991

E.12 Reparenting

Clients must be aware that some window managers will reparent their top-level windows, so that a window that was created as a child of the root or pseudo-root will be displayed as a child of a window belonging to the window manager. The effects that this reparenting will have on the client are:

- The parent value returned by a **QueryTree** request will no longer be the value supplied to the **CreateWindow** that created the reparented window. Normally, the client does not need to be aware of the identity of the window to which the top-level window has been reparented. In particular, a client wishing to create further top-level windows should continue to use the root or pseudo-root as their parent.

- The server interprets the *x,y* coordinates in a **ConfigureWindow** request in the new parent's coordinate space. Normally, these coordinates are not interpreted because a reparenting window manager usually has intercepted these operations (see "Redirecting Client Operations" in topic E.13).

- The *x,y* coordinates returned by a **GetGeometry** request are in the coordinate space of the parent and therefore not useful after a reparent operation.

A client requiring notification of when it is reparented can select **StructureNotify** on its top-level window to receive a **ReparentNotify** event if and when reparenting takes place.

If the window manager reparents the window of a client window the reparented window will be placed in the saveset of the parent window. This means that, if the window manager terminates, the client window is not destroyed and is remapped if unmapped.

When the window manager gives up control over a client top-level window, the window manager reparents it (and any associated windows, such as **WM_TRANSIENT_FOR** windows) back to the root.

¦ Copyright IBM Corp. 1985, 1991

E.13 Redirecting Client Operations

Clients must be aware that some window managers arrange for client requests to be intercepted and redirected. Redirected requests are not executed; they result, instead, in events being sent to the window manager to alter the arguments or to perform the request on behalf of the client.

The possibility that a request can be redirected means that a client can not assume that request that can be redirected is actually performed when the request is issued. For example, the sequence:

```
MapWindow A
PolyLine A GC <point> <point> ....
```

is incorrect, since the **MapWindow** request can be intercepted and the **PolyLine** output made to an unmapped window. The client must wait for an **Expose** event before drawing in the window. (2)

Another example is:

```
ConfigureWindow width=N height=M
<output assuming window is N by M>
```

which incorrectly assumes that the **ConfigureWindow** request is actually executed with the arguments supplied.

The requests which can be redirected are:

MapWindow

ConfigureWindow

CirculateWindow

A window with the override-redirect bit set is immune from redirection, but the bit should be set on top-level windows only in cases where other windows should be prevented from processing input while the override-redirect window is mapped (see "WM_TRANSIENT_FOR" in topic E.3.6).

(2) This is true even if the client set *backing-store* to **Always**. The *backing-store* value is only a hint, and the server can stop maintaining *backing-store* contents at any time.

¦ Copyright IBM Corp. 1985, 1991

E.14 Moving Windows

If the window manager moves a top-level window without changing its size, the client can elect to receive notification by setting the **WindowMoved** bit in the **WM_HINTS** structure. Notification is through a **ClientMessage** event whose *type* field is **WM_WINDOW_MOVED** and whose *data* field contains the new root or pseudo-root *X* and *Y*.

Clients must not respond to being moved by attempting to move themselves to a better location.

¦ Copyright IBM Corp. 1985, 1991

E.15 Resizing Windows

The client can receive notification of being resized by selecting **StructureNotify** on its top-level windows. A **ConfigureNotify** event on a top-level window implies that the window position on the root can have changed, even though its position in its parent is unchanged, because the window may have been reparented. Note that the coordinates in the event will not, in this case, be meaningful.

The response of the client to being resized should be to accept the size given. Clients must not respond to being resized by attempting to resize themselves to a better size. Clients should unmap themselves and go to the **Iconic** state.

¦ Copyright IBM Corp. 1985, 1991

E.16 Deiconifying Windows

If the top-level window has a **WM_HINTS** property and its *initial_state* field is not **IgnoreState**, the window will be in **Normal** state if it is mapped and in **Iconic** state if it is unmapped. This is true even if the window has been reparented; the window manager will unmap the window as well as its parent when switching to **Iconic** state.

The client can be notified of these state changes if **StructureNotify** is selected on the top-level window. An **UnmapNotify** event is received when it goes **Iconic** and an **MapNotify** event when it goes **Normal**.

¦ Copyright IBM Corp. 1985, 1991

E.17 Changing Colormaps

Clients can be notified that their colormaps are being installed or uninstalled if **ColormapNotify** is on their top-level windows. When the window is installed or uninstalled, clients receive **ColormapNotify** events with the *new* field **FALSE**.

Window managers install colormaps only in top-level windows. However, installation of subwindows might result from the window manager installation. Clients should install sub-window colormaps only when the top-level window already has its colormap installed.

There is a race condition; the **InstallColormap** request does not take a timestamp, and it can be executed after the top-level colormap has been uninstalled.

Clients that need to install their own colormaps and expect input from all their own colormaps, should install the colormaps only when they have input focus and the top-level colormap is installed. Clients that need to install their own colormaps, and never expect input, should install them only when their top-level window has its colormap installed.

¦ Copyright IBM Corp. 1985, 1991

F.0 Appendix F. Error Codes

These error codes can be returned by the **Xlib** and **FXlib** functions. One error code can be returned for more than one reason.

Error Code	Description
BadAccess	A client attempted to grab a key or button combination already grabbed by another client. A client attempted to free a colormap entry that it did not already allocate. A client attempted to store into a read-only colormap entry. A client attempted to modify the access control list from a host other than the local (or otherwise authorized) host. A client attempted to select an event type that another client has already selected and only one client can select at a time.
BadAlloc	The server failed to allocate the requested resource or server memory.
BadAtom	A value for an Atom argument does not name a defined Atom.
BadColor	A value for a Colormap argument does not name a defined Colormap.
BadCursor	A value for a Cursor argument does not name a defined Cursor.
BadDrawable	A value for a Drawable argument does not name a defined Window or Pixmap.
BadFont	A value for a Font or GC argument does not name a defined Font.
BadGC	A value for a GC argument does not name a defined GC.
BadIDChoice	The value chosen for a resource identifier is either not included in the range assigned to the client or 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. A server that generates this error for a core request is deficient. As such, this error is not listed for any of the requests. However, clients should be prepared to receive such errors and either handle or discard them.
BadLength	The length of a request is shorter or longer

Copyright IBM Corp. 1985, 1991

	than that minimally required to contain the arguments. This usually means an internal **Xlib** error.
BadMatch	In a graphics request, the root and depth of the GC does not match that of the drawable. An InputOnly window is used as a Drawable. Some argument or pair of arguments has the correct type and range but fails to match in some other way required by the request. An InputOnly window locks this attribute. The values do not exist for an InputOnly window.
BadName	A font or color of the specified name does not exist.
BadPixmap	A value for a Pixmap argument does not name a defined Pixmap.
BadRequest	The major or minor opcode does not specify a valid request.
BadValue	Some numeric value falls 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 for a Window argument does not name a defined Window.

Note: The **BadAtom**, **BadColor**, **BadCursor**, **BadDrawable**, **BadFont**, **BadGC**, **BadPixmap**, and **BadWindow** errors are also used when the argument type is extended by union with a set of fixed alternatives, for example, **Window** or **PointerRoot** or **None**.

Copyright IBM Corp. 1985, 1991

BACK_1 Glossary

access control list. A list of hosts (maintained by X-Windows) that have access to client programs. By default, only programs on the local host and those in this list, also known as *access list*, can use the display. The list can be changed by clients on the local host and by some server implementations can also modify the list. The authorization protocol name and data received by the server at connection setup may affect the list as well.

action table. A table that specifies the mapping of externally available procedure strings to the corresponding procedure implemented by the widget class. All widget class records contain an action table.

active grab. A grab actually owned by the grabbing client. See also *button grabbing*, *grab*, and *passive grab*.

ancestor. A widget that has inferior widgets. In other words, the superior or predecessor of an inferior widget. If *W* is inferior of *A*, then *A* is an ancestor of *W*.

atom. A unique ID corresponding to a string name. Atoms are used to identify properties, types, and selections.

backing store. The collection of off-screen saved pixels maintained by the server.

bit gravity. The attraction of window contents for a location in a window. When a window is resized, its contents can be relocated. The server can be requested to relocate the previous contents to a region of the window.

bit plane. On a color display, each pixel has more than one bit defined. Data in display memory can be either pixels (multiple bits per pixel) or bit planes. There is a 1-bit plane for each usable bit in the pixel.

bitmap. A pixmap with a depth of one bit plane.

bounding box. A geometry management technique. There are four types: fixed, heterogeneous, homogeneous, and shell.

button grabbing. Enacting a grab using a mouse button. This is an *active grab*.

byte order. The order of bytes as defined by the server for pixmap or bitmap data. Clients with different native byte ordering must swap bytes as necessary.

¦ Copyright IBM Corp. 1985, 1991

cache. A buffer storage that contains frequently-accessed instructions and data. A cache is used to reduce access time. See also *write back cache*.

child. A first-level subwindow.

children. The first-level subwindows. See also *managed children*.

class. A general group to which a specific object belongs. See also *widget class*.

client. An application program that connects to X-Windows server by an InterProcess Communication (IPC) path, such as a Transmission Control Protocol (TCP) connection or a shared memory buffer. The program can be referred to as the client of the server, but it is actually the IPC path itself. Programs with multiple paths open to the server are viewed as multiple clients by the protocol.

clip list. A list of rectangles designated for clipping.

clipping region. A type of graphics output. In a graphics context, the image defined by the bitmap or rectangles used to restrict output to a particular region of a window.

color cell. An entry in a colormap that consists of three values based on red, green, and blue intensities. The values are 16-bit, unsigned numbers. Zero represents the minimum intensity. The values are scaled by the server to match the particular display in use.

colormap. A set of color cells. A pixel value indexes the colormap to produce RGB-value intensities. A colormap consists of a set of entries defining color values that, when associated with a window, is used to display the contents of the window. Depending on hardware limitations, one or more colormaps can be installed at one time, such that windows associated with those maps display correct colors. The two classes of colormaps are *direct color* and *pseudocolor*.

composite widget. A widget that is a compilation of implementation-defined children, contains methods for managing the geometry of any widget, and usually has at least one normal child, but may have none. A subclass of Core widget.

connection. The IPC path between the server and a client program. A client program typically, but not necessarily, has one connection to the server over which requests and events are sent.

connection close. All events made by the client are discarded and the server resets its state to having no connections.

¦ Copyright IBM Corp. 1985, 1991

constant. A structure that is initialized at compile-time and never changed except for a one-time class initialization and an in-place compilation of resource lists. For example, the contents of a widget class are constant.

Constraint widget. A widget that is a subclass of a composite widget. It manages the geometry of its children based on constraints associated with each child.

containment. A pointer is *contained* if the pointer is located in the window, and not within an inferior of the window, and the cursor hotspot is within a visible region of a viewable window or one of its inferiors. The border of the window is considered part of the window.

coordinate system. A given convention for locating pixels on a given display or window, where *X* is the horizontal axis and *Y* is the vertical axis. The origin is [0,0] at the upper-left corner. For a window, the origin is at the upper-left, inside the border. Coordinates are discrete and specified in pixels. Each window and pixmap has its own coordinate system.

Core widget. The widget that contains the definitions of fields common to all widgets. All widgets are subclasses of the Core widget.

cursor. The visible shape of the pointer on a screen. A cursor consists of a hotspot, a source bitmap, and a pair of colors.

depth. (1) The number of bits per pixel for a window or pixmap. (2) For a GContext, the depth is the depth of the root of the GContext. (3) The depth of the drawables that can be used in conjunction with graphics output.

device. An input device, such as a keyboard, mouse, table, track-ball, or button box.

direct color. (1) A class of colormap in which a pixel value is decomposed into three separate subfields for indexing. One subfield indexes an array to produce red intensity values, the second indexes another array for blue intensity values, and the third for green intensity values. The RGB values can be changed dynamically. This is mutually exclusive to the **Pseudocolor** colormap color. (2) Also DirectColor, a value.

display. A set of one or more screens and input devices that are driven by a single X Server.

drawable. A collective term for both windows and pixmaps when used as destinations in graphics operations. However, an InputOnly window cannot be used as a source or destination drawable in a graphics operation.

¦ Copyright IBM Corp. 1985, 1991

dynamic. A style of creating pop-ups menus.

event. Information generated either asynchronously from a device or as the side-effect of a client request. Events are grouped into types and are not sent to a client by the server unless the client has issued a specific request for information of that type. Events are usually reported relative to a window.

event mask. The set of event types that a client requests relative to a window.

event synchronization. Allows synchronous processing of device events to avoid conflict when de-multiplexing device events to clients. Because mouse and keyboard events often occur almost simultaneously, event synchronization is important to window management operations.

event propagation. Device-related events propagate from the source window to ancestor windows until some client has expressed interest in handling that type of event or until the event is discarded explicitly.

event source. The smallest window containing the pointer is the *source* of a device-related event.

exposure event. An event sent to clients to inform them when contents have been lost, as when when windows are obscured or reconfigured. Servers do not guarantee the preservation of window contents when they are obscured or reconfigured.

extension. To extend the system, the named extensions can be defined for the Core protocol, including extensions to output requests, resources, and event types.

fixed box. (1) A geometry management technique. (2) A type of bounding box that has a fixed number of children created by the parent. These managed children do not make geometry manager requests.

focus window. See *input focus*.

font. A set of glyphs, usually characters. The protocol does not translate or interpret character sets. The client indicates values used to access the glyph arrays. A font contains additional metric information to determine inter-glyph and inter-line spacing.

frozen event. To change the screen, clients can freeze event processing.

geometry. The size of a widget is changed using geometry management

¦ Copyright IBM Corp. 1985, 1991

routines. Synonomous with layout.

glyph. An image, usually of a character, in a font.

grab. The act of selecting keyboard keys, the keyboard, pointer buttons, the pointer, and the server for exclusive use by a client. In general, these facilities are not intended to be used by normal applications, but are intended for various input and window managers to implement various styles of user interfaces. See also *active grab, passive grab, button grabbing, pointer grabbing,* and *key grabbing*.

graphics context (GC). The storage area for various kinds of graphics output, such as foreground pixel, background pixel, line width, and clipping region. Also known as *GC* and *GContext*, a graphics context can be used only with drawables that have the same root and the same depth as the graphics context.

gravity. The contents of windows or subwindows have an attraction to a location within the window. This determines how the window ID is resized. See also *bit gravity*, *widget gravity*, and *window gravity*.

gray scale. (1) A type of degenerate pseudocolor where the red, green, and blue values in any given colormap entry are equal, thus producing shades of gray. The gray values can be changed dynamically.

GC. See *graphics context*.

GC caching. Allows independent change requests to be merged into one protocol request.

GContext. See *graphics context*.

heterogeneous box. A geometry management technique where a type of bounding box can be resized and has a specific location where each child is placed. Instead of the location being defined by pixels, it is expressed in terms of the relationship between a child and the parent or between the child and other specific children. Usually a subclass of Constraint class.

homogeneous box. A geometry management technique where a type of bounding box treats all children equally and applies the same geometry constraints to each child.

hooking routines. Functions that connect with the library but remain outside the library; protocol extension procedures. Usually called *stubs*.

hotspot. The spot associated with a cursor that corresponds to the coordinates reported for the pointer. A cursor has an associated *hotspot*,

¦ Copyright IBM Corp. 1985, 1991

which defines a point in the cursor that corresponds to the coordinates reported for the pointer.

identifier. A unique value associated with a resource that a client program uses to name the resource. An identifier can be used over any connection to name the resource.

inherit. A child's resources that are copied from its parent.

inferiors. All the subwindows nested below a window.

input focus. A window defining the scope for processing keyboard input. By default, keyboard events are sent to the client using the window the pointer is in. It is also possible to attach the keyboard input to a specific window. Events are then sent to the appropriate client regardless of the pointer position. Synonymous with *focus window*.

input manager. A client that controls keyboard input and is usually part of a window manager.

InputOutput window. A kind of opaque window used for input and output. InputOutput windows can have both InputOutput and InputOnly windows as inferiors.

InputOnly window. An invisible window that can be used to control such things as cursors, input event generation, and grabbing. This window cannot be used for graphics requests.

instance. See *widget instance*.

InterProcess Communication (IPC). A communication path. See also *client*.

intermediate nodes. On the widget tree, widgets with one or more children.

International Standards Organization (ISO). An international body that standardizes goods and services. For X-Windows, standards relating to character sets and fonts.

Internet Protocol (IP). The protocol that provides the interface from the higher level host-to-host protocols to the local network protocols.

Intrinsics. A set of management mechanisms that provides for constructing and interfacing between composite widgets, their children, and other clients. Also, provides the ability to organize a collection of widgets into an application.

¦ Copyright IBM Corp. 1985, 1991

IP. See *Internet Protocol*.

IPC. See *InterProcess Communication*.

ISO. See *International Standards Organization*.

Japanese Industry Standard (JIS). A standard of coding character sets.

JIS. See *Japanese Industry Standard*.

key grabbing. Keys on the keyboard can be passively grabbed by a client. Or the keyboard can be actively grabbed by the client when a key is pressed.

keyboard grabbing. A client can actively grab control of the keyboard and key events will be sent to that client rather than the client to which the events would normally have been sent.

keysym. An encoding of a symbol on a keycap on a keyboard.

leaves. On a widget tree, widgets with no children.

managed children. Children whose managed field is True can have their layout (geometry) changed so that they can be repositioned and resized.

managed window. See *managed children*.

mapped. A window is said to be *mapped* if a map call has been performed on it. Contrast with an unmapped window and its inferiors, which are neither viewable nor visible.

mapping. A window on which a map call has been performed. Contrast with an unmapped window, which can be neither viewable nor visible.

MBCS. See MultiByte Character Set.

method. The functions or procedures that a widget itself implements.

modal pop-up. A window that normally is not visible to the window manager and available only after the manager is turned off. This pop-up disables user-event processing except for events that occur in the dialog box.

modeless pop-up. A window that is normally visible and is controlled by

¦ Copyright IBM Corp. 1985, 1991

the window manager.

modifier keys. Keys such as Shift, ShiftLock, Control, Alt, CapsLock, and Meta.

monochrome. A special case of static gray in which there are only two colormap entries.

MultiByte Character Set (MBCS). A set of encoding characters that accommodates the large character sets in which many oriental languages are written.

multiplex. To interleave or simultaneously transmit two or more messages on a single channel.

object. A software abstraction consisting of private data and private and public routines that operate on the private data. Users of the abstraction can interact with the object only through calls to the public routines of the objects.

obscure. (1) A state of being for a window. A window is obscured if another window is in front of it making the obscured window only *partially* viewable. Window *B* is obscured by window *A* if both are viewable InputOutput windows and *A* is higher in the global stacking order and the rectangle defined by the outside edges of *A* intersects the rectangle defined by the outside edges of *B*. (2) An action one window does to another when it *partially* obstructs the viewing of the other. Window *A* obscures window *B* if both are viewable InputOutput windows, *A* is higher in the global stacking order, and the rectangle defined by the outside edges of *A* intersects the rectangle defined by the outside edges of *B*.

occlude. (1) A state of being for a window. A window is occluded if the view of it is *completely* obstructed by another window. Window *B* is occluded by window *A* if both are mapped, *A* is higher in the global stacking order, and if no part of *B*'s border is viewable. (2) An action one window does to another when it *completely* obstructs the view of the other.

padding. Bytes inserted in the data stream to maintain alignment of the protocol requests on natural boundaries. Padding increases the ease of portability to some machine architectures.

paint. In computer graphics, to shade an area of a display image.

parent window. The window that controls the size and location of its children. If a window has children, it is a parent window.

passive grab. Grabbing a key or button is a passive grab. The grab becomes an active grab when the key or button is actually pressed.

¦ Copyright IBM Corp. 1985, 1991

pixel value. The number of bit planes used in a particular window or pixmap. For a window, a *pixel* value indexes a colormap and derives an actual color to be displayed. A pixel is an *N*-bit value, where *N* is the number of bit planes (the depth) used in a particular window or pixmap.

pixmap. A three-dimensional array of bits. A pixmap can be thought of as a two-dimensional array of pixels, with each pixel being a value from zero to 2(*N - 1*), with *N* as the depth (Z-axis) of the pixmap.

plane. When a pixmap or window is thought of as a stack of bitmaps, each bitmap is called a *plane* or *bit plane*.

plane mask. A bit mask restricting graphics operations to affect a subset of bit planes. It is stored in a graphics context. Graphics operations can be restricted to affect only a subset of bit planes of a destination.

pointer. The device attached to the cursor and tracked on the screen.

pointer grabbing. A client can actively grab control of the pointer so that button and motion events will be sent to that client rather than the client to which the events normally would have been sent.

pointing device. A device with effective dimensional motion, usually a mouse. One visible cursor is defined by the Core protocol, and it tracks whatever pointing device is attached as the pointer.

pop-down. An action referring to a type of widget that closes when a pointer button is released.

pop-up. An action referring to a type of widget that opens automatically when a pointer button is held down within certain windows.

pop-up cascade. Several spring-loaded pop-ups emanating in succession from one modal pop-up.

pop-up child. A child on the pop-up list.

pop-up list. The proper place in the widget hierarchy for a pop-up to get resources.

pop-up widget. A window child of the root that is attached to its widget parent differently from the normal widget; not geometrically constrained by its parent widget.

primitive widget. (1) A widget that instantiates its own children of a

¦ Copyright IBM Corp. 1985, 1991

known class rather than those instantiated by external clients. (2) A widget that has no geometry management methods. Responsible for operations requiring downward tranversal below themselves.

property. The name, type, data format, and data associated with a window. By using properties, clients and a window manager share information, such as resize hints, program names, and icon formats. It is a general-purpose naming mechanism for clients. The protocol does not interpret properties.

property list. The list of properties that are defined for a particular window.

pseudocolor. (1) A class of colormap in which a pixel value indexes the colormap entry to produce independent red, green, and blue values. That is, the colormap is viewed as an array of triples (RGB values). The RGB values can be changed dynamically. This is mutually exclusive to the direct color colormap class. (2) Also PseudoColor, a value.

quark. Synonymous with string.

raise. To make the stacking order of a window higher.

rectangle. A rectangle specified by **[x,y,w,h]** has an infinitely thin outline path with corners at **[x,y]**, **[x+w,y]**, **[x+w,y+h]** and **[x,y+h]**. When a rectangle is filled, the lower-right edges are not drawn. For example, if *w*=*h*=0, nothing would be drawn; if *w*=*h*=1, a single pixel would be drawn.

redirecting control. Transferring an operation to a specified client. Used when window managers or client programs enforce window layout policy to prevent attempts to change the size or position of a window.

region. An area within a bitmap, a pixmap, a screen, or a window.

reply. The way information requested by a client program is sent back to the client. Both events and replies are multiplexed on the same connection. Most requests do not generate replies; some generate multiple replies.

request. A command to the server to send a single block of data over a connection.

required list. An ordered list containing a subset of the installed colormaps.

resource. (1) Items such as windows, pixmaps, cursors, fonts, graphics contexts, and colormaps are known as resources. Each has a unique identifier associated with it for naming purposes. The lifetime of a resource is bounded by the lifetime of the connection over which the

¦ Copyright IBM Corp. 1985, 1991

resource was created. (2) A named piece of data in a widget that can be set by a client, by an application, or by user defaults.

RGB value. Red, green, and blue (RGB) intensity values are used to define a color. These values are always represented as 16-bit unsigned numbers with zero, the minimum intensity, and 65535, the maximum intensity. The X Sserver scales these values to match the display hardware.

root. (1) The screen on which the window is created. The root of a pixmap or GContext is the same as the root of the drawable used when the pixmap or GContext was created. The root of a pixmap or graphics context is the same as the root of whatever drawable was used when the pixmap or graphics context was created. The root of a window is the root window under which the window was created. (2) On the widget tree, the Shell widget returned by XtInitialize or XtCreateApplicationShell.

root window. Each screen has a root window covering it. This window cannot be reconfigured or unmapped, but otherwise performs like any other window. A root window has no parent.

rubber-band outline. A window with a moveable outline.

saveset. A list of window clients that should not be destroyed when a connection is closed and should be remapped or unmapped. Usually used by window managers to avoid lost windows if the manager is ended abnormally.

scanline. A list of pixel or bit values viewed as a horizontal row (all values have the same *y* coordinate). The values are ordered by increasing the *x* coordinate. As part of an image, the next scanline is ordered by increasing the *y* coordinate.

scanline order. An image represented by scanlines ordered by increasing the *y* coordinate.

screen. A server can provide several independent screens that typically have physically independent monitors (display screens). This is the expected configuration when there is only a single keyboard and pointer shared among the screens. A Screen structure contains the information about that screen and is linked to the Display structure.

selection. An indirect property of a dynamic type maintained by the client (the owner) but belonging to the user. It is not private to a particular window subhierarchy or a particular set of clients. When a client asks for the contents of a selection, it specifies a target type. This target type can be used to control the transmitted representation of the contents.

server. Provides the basic windowing mechanism. It handles IPC connections from clients, de-multiplexes graphics requests onto screens, and multiplexes input back to clients.

¦ Copyright IBM Corp. 1985, 1991

server grabbing. When a client seizes the server for exclusive use to prevent processing requests from other client connections until the grab is complete. This is typically a transient state for such things as rubber-banding and pop-up menus or to execute requests indivisibly.

shell box. A geometry management technique where a type of bounding box can have only one child that is exactly the same size as the shell.

Shell widget. Holds the top-level widgets that communicate directly with the window manager. These widgets do not have parents.

sibling. Children of the same parent window.

spring-loaded pop-up. A kind of widget, such as a menu, that is not visible to the window manager. The spring-loaded pop-up disables user-event processing except for events that occur in the menu.

stacking order. The relationship between sibling windows that stack on top of each other.

static. A style of creating pop-ups.

static color. (1) Static color can be viewed as a degenerate case of pseudocolor in which the RGB values are predefined and read-only. See *pseudocolor*. (2) Also StaticColor, a value.

static gray. (1) A degenerate case of gray scale in which the gray values are predefined and read-only. The values are typically near-linear increasing ramps. See *gray scale* and *monochrome*. (2) Also StaticGray, a value.

stipple. A bitmap used to tile a region. A stipple pattern serves as an additional clip mask for a fill operation with the foreground color.

stubs. Hooking functions used as extensions to the protocol to generate protocol requests for X-Windows.

TCP. See *Transmission Control Protocol*.

tile. (1) A bitmap. (2) To fill a region with a bitmap. To replicate a pixmap in two dimensions.

timestamp. A time value expressed in milliseconds, typically since the last server reset. Timestamp values wrap around usually after 49.7 days. The server, once given its current time, is represented by *timestamp T* and

¦ Copyright IBM Corp. 1985, 1991

always interprets timestamps from clients by treating half of the timestamp space as being earlier in time than *T*, and half the timestamp space as being later in time than *T*. One *timestamp* value, represented by the constant CurrentTime, is never generated by the server. This value is reserved for use in requests to represent the current server time.

translation table. (1) A table that specifies the mapping of events or event sequences to procedure names. (2) A string containing a list translating the events to procedure calls.

Transmission Control Protocol (TCP). A communications protocol used in Internet and any other network following the U.S. Department of Defense standards for inter-network protocol. Provides a reliable host-to-host protocol in packet-switched communications networks and in an interconnnected system of such networks. It assumes that Internet Protocol is the underlying protocol.

true color. (1) A degenerate case of direct color in which the subfields in the pixel value directly encode the corresponding RGB values. That is, the colormap has predefined read-only RGB values. The values are typically near-linear increasing ramps. (2) Also TrueColor, a value.

type. An arbitrary atom used to identify the data. A type is solely for the benefit of clients and is not interpreted by the server. X-Windows predefines type atoms for many frequently-used types. Clients also can define new types.

unmanaged window. A window whose size cannot be changed.

unviewable. Pertaining to a mapped window with an unmapped ancestor.

user. A person interacting with a workstation and X-Windows.

valuator. Synonymous with dial.

viewable. Pertaining to a mapped window whose ancestors are all mapped; not necessarily visible. Graphics requests can be performed on a window when it is not viewable, but output will not be retained unless the server is maintaining backing store.

visible. A region of a window that is viewable and not occluded on the screen by the user.

widget. (1) The fundamental data type of the X-Windows Toolkit. (2) An object providing a user-interface abstraction; for example, a Scrollbar widget. It is the combination of an X-Windows window (or subwindow) and its associated semantics. Logically, it is a rectangle with associated input and output semantics, although some can be input-only or output-only. Each widget belongs to one widget class. A widget

¦ Copyright IBM Corp. 1985, 1991

implements procedures through its widget class structure. See also *composite widget*, *Core widget*, *primitive widget*, and *Shell widget*.

widget class. The general group that a specific widget belongs to, otherwise known as the widget type. Physically, it is a pointer to a structure.

widget gravity. Synonymous with *window gravity*.

widget instance. A specific widget object as opposed to a general widget class. It is composed of a data structure containing instance-specific values and another data structure containing information applicable to all widgets of that class.

widget programmer. A programmmer who adds new widgets to the X-Windows Toolkit.

widget tree. The symbolic structure for X-Windows Toolkit code. The basic element is a widget class. See also *leaves*, *intermediate nodes*, and *root*.

widget type. Synonymous with *widget class*.

window gravity. The attraction of a subwindow to some part of its parent. Window gravity causes subwindows to be automatically repositioned, relative to an edge, corner, or center of a window when resized.

window manager. The client that manipulates windows on a screen and provides much of the user interface.

write back cache. GCs cached by the library to allow merging independent change requests into one protocol request.

X-Windows Toolkit. A collection of basic functions for developing a variety of application environments. Toolkit functions manage Toolkit initialization, widgets, memory, events, geometry, input focus, selections, resources, translation of events, graphics contexts, pixmaps, and errors.

X Server. Synonymous with *server*.

XYFormat. The format of a pixmap organized as a set of bitmaps representing individual bit planes that appear in most-significant to least-significant bit order.

ZFormat. The format of a pixmap organized as a set of pixel values in scanline order.

¦ Copyright IBM Corp. 1985, 1991

Special Characters

A

¦ Copyright IBM Corp. 1985, 1991

¦ Copyright IBM Corp. 1985, 1991

B

¦ Copyright IBM Corp. 1985, 1991

¦ Copyright IBM Corp. 1985, 1991

¦ Copyright IBM Corp. 1985, 1991

¦ Copyright IBM Corp. 1985, 1991

¦ Copyright IBM Corp. 1985, 1991

¦ Copyright IBM Corp. 1985, 1991

¦ Copyright IBM Corp. 1985, 1991

D

¦ Copyright IBM Corp. 1985, 1991

¦ Copyright IBM Corp. 1985, 1991

¦ Copyright IBM Corp. 1985, 1991

E

¦ Copyright IBM Corp. 1985, 1991

¦ Copyright IBM Corp. 1985, 1991

¦ Copyright IBM Corp. 1985, 1991

F

¦ Copyright IBM Corp. 1985, 1991

¦ Copyright IBM Corp. 1985, 1991

¦ Copyright IBM Corp. 1985, 1991

G

¦ Copyright IBM Corp. 1985, 1991

¦ Copyright IBM Corp. 1985, 1991

H

¦ Copyright IBM Corp. 1985, 1991

I

¦ Copyright IBM Corp. 1985, 1991

¦ Copyright IBM Corp. 1985, 1991

¦ Copyright IBM Corp. 1985, 1991

¦ Copyright IBM Corp. 1985, 1991

¦ Copyright IBM Corp. 1985, 1991

N

¦ Copyright IBM Corp. 1985, 1991

¦ Copyright IBM Corp. 1985, 1991

P

¦ Copyright IBM Corp. 1985, 1991

¦ Copyright IBM Corp. 1985, 1991

¦ Copyright IBM Corp. 1985, 1991

Q

¦ Copyright IBM Corp. 1985, 1991

R

¦ Copyright IBM Corp. 1985, 1991

¦ Copyright IBM Corp. 1985, 1991

¦ Copyright IBM Corp. 1985, 1991

¦ Copyright IBM Corp. 1985, 1991

¦ Copyright IBM Corp. 1985, 1991

¦ Copyright IBM Corp. 1985, 1991

¦ Copyright IBM Corp. 1985, 1991

¦ Copyright IBM Corp. 1985, 1991

T

¦ Copyright IBM Corp. 1985, 1991

U

¦ Copyright IBM Corp. 1985, 1991

¦ Copyright IBM Corp. 1985, 1991

¦ Copyright IBM Corp. 1985, 1991

X

¦ Copyright IBM Corp. 1985, 1991

¦ Copyright IBM Corp. 1985, 1991

¦ Copyright IBM Corp. 1985, 1991

¦ Copyright IBM Corp. 1985, 1991

¦ Copyright IBM Corp. 1985, 1991

¦ Copyright IBM Corp. 1985, 1991

¦ Copyright IBM Corp. 1985, 1991

¦ Copyright IBM Corp. 1985, 1991

¦ Copyright IBM Corp. 1985, 1991

¦ Copyright IBM Corp. 1985, 1991

¦ Copyright IBM Corp. 1985, 1991