AIX Version 3 for RISC System/6000™

Calls and Subroutines Reference: User Interface

Volume 4

SC23-2198-00

First Edition (March 1990)

This edition of the *AIX Calls and Subroutines Reference for IBM RISC System/6000* applies to IBM AIX Version 3 for RISC System/6000, Version 3 of IBM AIXwindows Environment/6000, IBM AIX System Network Architecture Services/6000, IBM AIX 3270 Host Connection Program/6000, IBM AIX 3278/79 Emulation/6000, IBM AIX Network Management/6000, and IBM AIX Personal Computer Simulator/6000 and to all subsequent releases of these products until otherwise indicated in new releases or technical newsletters.

The following paragraph does not apply to the United Kingdom or any country where such provisions are inconsistent with local law: INTERNATIONAL BUSINESS MACHINES CORPORATION PROVIDES THIS MANUAL "AS IS" WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESSED OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE. Some states do not allow disclaimer of express or implied warranties in certain transactions; therefore, this statement may not apply to you.

IBM does not warrant that the contents of this publication or the accompanying source code examples, whether individually or as one or more groups, will meet your requirements or that the publication or the accompanying source code examples are error–free.

This publication could include technical inaccuracies or typographical errors. Changes are periodically made to the information herein; these changes will be incorporated in new editions of the publication. IBM may make improvements and/or changes in the product(s) and/or the program(s) described in this publication at any time.

It is possible that this publication may contain reference to, or information about, IBM products (machines and programs), programming, or services that are not announced in your country. Such references or information must not be construed to mean that IBM intends to announce such IBM products, programming, or services in your country. Any reference to an IBM licensed program in this publication is not intended to state or imply that you can use only IBM's licensed program. You can use any functionally equivalent program instead.

Requests for copies of this publication and for technical information about IBM products should be made to your IBM Authorized Dealer or your IBM Marketing Representative.

A reader's comment form is provided at the back of this publication. If the form has been removed, address comments to IBM Corporation, Department 997, 11400 Burnet Road, Austin, Texas 78758–3493. IBM may use or distribute whatever information you supply in any way it believes appropriate without incurring any obligation to you.

Trademarks and Acknowledgements

The following trademarks and acknowledgements apply to this information:

AIX is a trademark of International Business Machines Corporation.

AIX/RT is a trademark of International Business Machines Corporation.

AIXwindows is a trademark of International Business Machines Corporation.

HP is a trademark of Hewlett Packard Inc.

HP–GL is a trademark of Hewlett–Packard Company.

IBM is a registered trademark of International Business Machines Corporation.

Operating System/2 and OS/2 are trademarks of International Business Machines Corporation.

OSF and OSF/Motif are trademarks of Open Software Foundation, Inc.

PAL is a trademark of International Business Machines Corporation.

Personal Computer AT and AT are trademarks of International Business Machines Corporation.

RISC System/6000 is a trademark of International Business Machines Corporation.

RT is a trademark of International Business Machines Corporation.

UNIX was developed and licensed by AT&T and is a registered trademark of AT&T Corporation.

Xstation Manager is a trademark of International Business Machines Corporation.

X Window System is a trademark of Massachusetts Institute of Technology.

X/OPEN is a trademark of X/OPEN Company Limited.

About This Book

This book provides information on AIXwindows classes, subroutines, and resource sets; Enhanced X–Windows subroutines, events, extensions, protocols and toolkit subroutines, and Curses and Extended Curses for use on the Advanced Interactive Executive Operating System (referred to in this text as AIX) for use on the IBM RISC System/6000.

This book is part of *AIX Calls and Subroutines Reference for IBM RISC System/6000*, SC23–2198. *AIX Calls and Subroutines Reference* is divided into the following four major sections:

- Volumes 1 and 2, *Calls and Subroutines Reference*: *Base Operating System*, contains reference information about the system calls, subroutines, functions, macros, and statements associated with AIX base operating system runtime services, communications services, and device services.
- Volumes 3 and 4, *Calls and Subroutines Reference*: *User Interface*, contain reference information about the AIXwindows widget classes, subroutines, and resource sets; the AIXwindows Desktop resource sets; the Enhanced X–Windows subroutines, macros, protocols, extensions, and events; the X–Window toolkit subroutines and macros; and the curses and extended curses subroutine libraries.
- Volume 5, *Calls and Subroutines Reference*: *Kernel Reference*, contains reference information about kernel services, device driver operations, file system operations subroutines, the configuration subsystem, the communications subsystem, the high function terminal (HFT) subsystem, the logical volume subsystem, the printer subsystem, and the SCSI subsystem.
- Volumes 6, *Calls and Subroutines Reference*: *Graphics*, contains reference information and example programs for the Graphics Library (GL) and the AIXwindows Graphics Support Library (XGSL) subroutines.

Who Should Use This Book

This book is intended for experienced programmers who understand the basic functions of the IBM RISC System/6000. To use this book effectively, you should be familiar with AIX or UNIX System V commands and subroutines, AIXwindows subroutines, and Enhanced X–Windows subroutines. If you are not already familiar with AIX or UNIX System V, refer to *AIX General Concepts and Procedures*.

How to Use This Book

Overview of Contents

This book contains the following alphabetically arranged sections on AIXwindows, Enhanced X–Windows, Curses and Extended Curses.

- **AIXwindows**
  - Classes
  - Subroutines
  - Resource Sets
  - Desktop Resource Sets
  - Window Management
- **Enhanced X–Windows**
  - Subroutines
  - Toolkit Subroutines

  - Protocols
  - Extensions
  - Events
- **Curses**
- **Extended Curses**

Highlighting

The following highlighting conventions are used in this book:

Bold	Identifies commands, keywords, files, directories, and other items whose names are predefined by the system.
Italics	Identifies parameters whose actual names or values are to be supplied by the user.
`Monospace`	Identifies examples of specific data values, examples of text similar to what you might see displayed, examples of portions of program code similar to what you might write as a programmer, messages from the system, or information you should actually type.

Related Publications

The following books contain information about or related to application programming interfaces:

- *AIX General Programming Concepts for IBM RISC System/6000*, Order Number SC23–2205.
- *AIX Communication Programming Concepts for IBM RISC System/6000*, Order Number SC23–2206.
- *AIX Kernel Extensions and Device Support Programming Concepts for IBM RISC System/6000*, Order Number SC23–2207.
- *AIX Files Reference for IBM RISC System/6000*, Order Number SC23–2200.
- *AIX User Interface Programming Concepts for IBM RISC System/6000*, Order Number SC23–2209.
- *IBM RISC System/6000 Problem Solving Guide*, Order Number SC23–2204.
- *XL C Language Reference for IBM AIX Version 3 for RISC System/6000*, Order Number SC09–1260.
- *XL C User's Guide for IBM AIX Version 3 for RISC System/6000*, Order Number SC09–1259.

Ordering Additional Copies of This Book

To order additional copies of this book, use Order Number SC23–2198.

Contents

Enhanced X–Windows Subroutines

AllPlanes Macro

Purpose

Returns the depth of the root window.

Libraries

Enhanced X-Windows Library (**libX11.a**)

FORTRAN 77 Library (**libXfx.a**)

Macro Syntax

AllPlanes()

C Syntax

unsigned long XAllPlanes()

FORTRAN Syntax

integer*4 fxallplanes
external fxallplanes
integer*4 *ReturnCode*
ReturnCode **= fxallplanes( )**

Description

Both the macro and the subroutine return a value with all bits set to 1, suitable for use in a plane argument to a procedure.

Macro information can be found in the **/usr/include/X11/Xlib.h** file.

Error Code

BadImplementation

Implementation Specifics

This Xlib subroutine is part of AIXwindows Run Time Environment in AIXwindows Environment/6000.

BitmapBitOrder Macro

Purpose

Returns the ordering of bits in a bitmap.

Libraries

Enhanced X-Windows Library (**libX11.a**)

FORTRAN 77 Library (**libXfx.a**)

Macro Syntax

BitmapBitOrder(*DisplayPtr)*

C Syntax

int XBitmapBitOrder(*DisplayPtr)*
**Display ****DisplayPtr;*

FORTRAN Syntax

integer*4 fxbitmapbitorder
external fxbitmapbitorder
integer*4 *DisplayPtr*
integer*4 *ReturnCode*
ReturnCode = **fxbitmapbitorder(***DisplayPtr***)**

Description

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.

Macro information can be found in the **/usr/include/X11/Xlib.h** file.

Parameter

DisplayPtr Specifies the connection to the X Server.

Return Values

LSBFirst Indicates least significant bit first.

MSBFirst Indicates most significant bit first.

Error Code

BadImplementation

Implementation Specifics

This Xlib subroutine is part of AIXwindows Run Time Environment in AIXwindows Environment/6000.

XBitmapPad Subroutine

Purpose

Returns the scanline pad unit of the server.

Libraries

Enhanced X-Windows Library (**libX11.a**)

FORTRAN 77 Library (**libXfx.a**)

Macro Syntax

BitmapPad(*DisplayPtr)*

C Syntax

int XBitmapPad(*DisplayPtr)*
**Display ****DisplayPtr*;

FORTRAN Syntax

integer*4 fxbitmappad
external fxbitmappad
integer*4 *DisplayPtr*
integer*4 *ReturnCode*
ReturnCode = **fxbitmappad(***DisplayPtr***)**

Description

Each scanline must be padded to a multiple of the bits returned by this macro or subroutine.

Macro information can be found in the **/usr/include/X11/Xlib.h** file.

Parameter

DisplayPtr Specifies the connection to the X Server.

Error Code

BadImplementation

Implementation Specifics

This Xlib subroutine is part of AIXwindows Run Time Environment in AIXwindows Environment/6000.

BitmapUnit Macro

Purpose

Returns the size of a bitmap unit.

Libraries

Enhanced X-Windows Library (**libX11.a**)

FORTRAN 77 Library (**libXfx.a**)

Macro Syntax

BitmapUnit(*DisplayPtr)*

C Syntax

int XBitmapUnit(*DisplayPtr)*
**Display ****DisplayPtr;*

FORTRAN Syntax

integer*4 fxbitmapunit
external fxbitmapunit
integer*4 *DisplayPtr*
integer*4 *ReturnCode*
ReturnCode = **fxbitmapunit(***DisplayPtr***)**

Description

Both the macro and the subroutine return the size of a bitmap unit in bits. The scanline is calculated in multiples of this value.

Macro information can be found in the **/usr/include/X11/Xlib.h** file.

Parameter

DisplayPtr Specifies the connection to the X Server.

Error Code

BadImplementation

Implementation Specifics

This Xlib subroutine is part of AIXwindows Run Time Environment in AIXwindows Environment/6000.

BlackPixel Macro

Purpose

Returns the black pixel value.

Libraries

Enhanced X-Windows Library (**libX11.a**)

FORTRAN 77 Library (**libXfx.a**)

Macro Syntax

BlackPixel(*DisplayPtr*, *ScreenNumber*)

C Syntax

unsigned long XBlackPixel(*DisplayPtr*, *ScreenNumber*)
Display **DisplayPtr*;
int *ScreenNumber;*

FORTRAN Syntax

integer*4 fxblackpixel
external fxblackpixel
integer*4 *DisplayPtr*
integer*4 *ScreenNumber*
integer*4 *ReturnCode*
ReturnCode = **fxblackpixel**(*DisplayPtr*, *ScreenNumber*)

Description

Both the macro and the subroutine return the black pixel value for the screen specified.

Macro information can be found in the **/usr/include/X11/Xlib.h** file.

Parameters

DisplayPtr	Specifies the connection to the X Server.
ScreenNumber	Specifies the screen number of the display device.

Error Code

BadImplementation

Implementation Specifics

This Xlib subroutine is part of AIXwindows Run Time Environment in AIXwindows Environment/6000.

BlackPixelOfScreen Macro

Purpose

Returns the black pixel value.

Libraries

Enhanced X-Windows Library (**libX11.a**)

FORTRAN 77 Library (**libXfx.a**)

Macro Syntax

BlackPixelOfScreen(*ScreenPtr***)**

C Syntax

unsigned long XBlackPixelOfScreen(*ScreenPtr)*
**Screen ****ScreenPtr;*

FORTRAN Syntax

integer*4 fxblackpixelofscreen
external fxblackpixelofscreen
integer*4 *ScreenPtr*
integer*4 *ReturnCode*
ReturnCode = **fxblackpixelofscreen(***ScreenPtr***)**

Description

Both the macro and the subroutine return the black pixel value of the screen specified.

Macro information can be found in the **/usr/include/X11/Xlib.h** file.

Parameter

ScreenPtr Specifies the screen of the display device.

Error Code

BadImplementation

Implementation Specifics

This Xlib subroutine is part of AIXwindows Run Time Environment in AIXwindows Environment/6000.

CellsOfScreen Macro

Purpose

Returns the number of colormap cells in the default colormap of the specified screen.

Libraries

Enhanced X-Windows Library (**libX11.a**)

FORTRAN 77 Library (**libXfx.a**)

Macro Syntax

CellsOfScreen(*ScreenPtr***)**

C Syntax

int XCellsOfScreen(*ScreenPtr)*
**Screen ****ScreenPtr;*

FORTRAN Syntax

integer*4 fxcellsofscreen
external fxcellsofscreen
integer*4 *ScreenPtr*
integer*4 *ReturnCode*
ReturnCode **= fxcellsofscreen(***ScreenPtr***)**

Description

Both the macro and the subroutine return the number of colormap cells in the default colormap for the screen specified.

Macro information can be found in the **/usr/include/X11/Xlib.h** file.

Parameter

ScreenPtr Specifies the screen of the display device.

Error Code

BadImplementation

Implementation Specifics

This Xlib subroutine is part of AIXwindows Run Time Environment in AIXwindows Environment/6000.

ConnectionNumber Macro

Purpose

Returns the file descriptor of the connection.

Libraries

Enhanced X-Windows Library (**libX11.a**)

FORTRAN 77 Library (**libXfx.a**)

Macro Syntax

ConnectionNumber(*DisplayPtr)*

C Syntax

int XConnectionNumber(*DisplayPtr)*
Display **DisplayPtr;*

FORTRAN Syntax

integer*4 fxconnectionnumber
external fxconnectionnumber
integer*4 *DisplayPtr*
integer*4 *ReturnCode*
ReturnCode = **fxconnectionnumber(***DisplayPtr*)

Description

Both the macro and the subroutine return a connection number, which is the file descriptor of the connection, for the specified display device.

Macro information can be found in the **/usr/include/X11/Xlib.h** file.

Parameter

DisplayPtr Specifies the connection to the X Server.

Error Code

BadImplementation

Implementation Specifics

This Xlib subroutine is part of AIXwindows Run Time Environment in AIXwindows Environment/6000.

DefaultColormap Macro

Purpose

Returns the default colormap ID.

Libraries

Enhanced X-Windows Library (**libX.a**)

FORTRAN 77 Library (**libXfx.a**)

Macro Syntax

DefaultColormap(*DisplayPtr, ScreenNumber*)

C Syntax

Colormap XDefaultColormap(*DisplayPtr*, *ScreenNumber*)
Display **DisplayPtr*;
int *ScreenNumber*;

FORTRAN Syntax

integer*4 fxdefaultcolormap
external fxdefaultcolormap
integer*4 *DisplayPtr*
integer*4 *ScreenNumber*
integer*4 *ReturnCode*
ReturnCode = **fxdefaultcolormap**(*DisplayPtr*, *ScreenNumber*)

Description

Both the macro and the subroutine return the default colormap ID for allocation on the specified screen. This colormap should be used for most routine color allocations.

Macros can be found in the **/usr/include/X11/Xlib.h** file.

Parameters

DisplayPtr	Specifies the connection to the X Server.
ScreenNumber	Specifies the screen number of the display device.

Error Code

BadImplementation

Implementation Specifics

This Xlib subroutine is part of AIXwindows Run Time Environment in AIXwindows Environment/6000.

DefaultColormapOfScreen Macro

Purpose

Returns the default colormap.

Libraries

Enhanced X-Windows Library (**libX11.a**)

FORTRAN 77 Library (**libXfx.a**)

Macro Syntax

DefaultColormapOfScreen(*ScreenPtr***)**

C Syntax

Colormap XDefaultColormapOfScreen(*ScreenPtr***)**
Screen ****ScreenPtr*;**

FORTRAN Syntax

integer*4 fxdefaultcolormapofscreen
external fxdefaultcolormapofscreen
integer*4 *ScreenPtr*
integer*4 *ReturnCode*
ReturnCode = **fxdefaultcolormapofscreen(***ScreenPtr***)**

Description

Both the macro and the subroutine return the default colormap of the screen specified.

Macros can be found in the **/usr/include/X11/Xlib.h** file.

Parameter

ScreenPtr Specifies the screen of the display device.

Error Code

BadImplementation

Implementation Specifics

This Xlib subroutine is part of AIXwindows Run Time Environment in AIXwindows Environment/6000.

DefaultDepth Macro

Purpose

Returns the depth (number of planes) of the default root window.

Libraries

Enhanced X-Windows Library (**libX11.a**)

FORTRAN 77 Library (**libXfx.a**)

Macro Syntax

DefaultDepth(*DisplayPtr*, *ScreenNumber*)

C Syntax

int XDefaultDepth(*DisplayPt*, *ScreenNumber*)
Display **DisplayPtr*;
int *ScreenNumber*;

FORTRAN Syntax

integer*4 fxdefaultdepth
external fxdefaultdepth
integer*4 *DisplayPtr*
integer*4 *ScreenNumber*
integer*4 *ReturnCode*
ReturnCode = **fxdefaultdepth**(*DisplayPtr*, *ScreenNumber*)

Description

Both the macro and the subroutine return the depth (number of planes) of the default root window for the specified screen. Other depths may also be supported on this screen.

Macros can be found in the **/usr/include/X11/Xlib.h** file.

Parameters

DisplayPtr	Specifies the connection to the X Server.
ScreenNumber	Specifies the screen number of the display device.

Error Code

BadImplementation

Implementation Specifics

This Xlib subroutine is part of AIXwindows Run Time Environment in AIXwindows Environment/6000.

DefaultDepthOfScreen Macro

Purpose

Returns the default depth (number of planes).

Libraries

Enhanced X-Windows Library (**libX11.a**)

FORTRAN 77 Library (**libXfx.a**)

Macro Syntax

DefaultDepthOfScreen(*ScreenPtr***)**

C Syntax

int XDefaultDepthOfScreen(*ScreenPtr)*
**Screen ****ScreenPtr;*

FORTRAN Syntax

integer*4 fxdefaultdepthofscreen
external fxdefaultdepthofscreen
integer*4 *ScreenPtr*
integer*4 *ReturnCode*
ReturnCode **= fxdefaultdepthofscreen(***ScreenPtr***)**

Description

Both the macro and the subroutine return the default depth (number of planes) of the specified screen.

Macros can be found in the **/usr/include/X11/Xlib.h** file.

Parameter

ScreenPtr Specifies the screen of the display device.

Error Code

BadImplementation

Implementation Specifics

This Xlib subroutine is part of AIXwindows Run Time Environment in AIXwindows Environment/6000.

DefaultGC Macro

Purpose

Returns the default graphics context (GC) of the default root window.

Libraries

Enhanced X-Windows Library (**libX11.a**)

FORTRAN 77 Library (**libXfx.a**)

Macro Syntax

DefaultGC(*DisplayPtr*, *ScreenNumber***)**

C Syntax

GC XDefaultGC(*DisplayPtr*, *ScreenNumber***)**
**Display ****DisplayPtr*;
int *ScreenNumber*;

FORTRAN Syntax

integer*4 fxdefaultgc
external fxdefaultgc
integer*4 *DisplayPtr*
integer*4 *ScreenNumber*
integer*4 *DefaultGraphicsContext*
DefaultGraphicsContext **= fxdefaultgc(***DisplayPtr*, *ScreenNumber***)**

Description

Both the macro and the subroutine return the default graphics context (GC) of the default root window for the specified screen. 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 as it is not used in any Xlib function. This GC should never be freed.

Macros can be found in the **/usr/include/X11/Xlib.h** file.

Parameters

DisplayPtr	Specifies the connection to the X Server.
ScreenNumber	Specifies the screen number of the display device.

Error Code

BadImplementation

Implementation Specifics

This Xlib subroutine is part of AIXwindows Run Time Environment in AIXwindows Environment/6000.

DefaultGCOfScreen Macro

Purpose

Returns the default graphics context (GC).

Libraries

Enhanced X-Windows Library (**libX11.a**)

FORTRAN 77 Library (**libXfx.a**)

Macro Syntax

DefaultGCOfScreen(*ScreenPtr***)**

C Syntax

GC XDefaultGCOfScreen(*ScreenPtr)*
**Screen ****ScreenPtr*;

FORTRAN Syntax

integer*4 fxdefaultgcofscreen
external fxdefaultgcofscreen
integer*4 *ScreenPtr*
integer*4 *ReturnCode*
ReturnCode **= fxdefaultgcofscreen(***ScreenPtr***)**

Description

Both the macro and the subroutine return the default graphics context (GC) of the specified screen.

Macros can be found in the **/usr/include/X11/Xlib.h** file.

Parameter

ScreenPtr Specifies the screen of the display device.

Error Code

BadImplementation

Implementation Specifics

This Xlib subroutine is part of AIXwindows Run Time Environment in AIXwindows Environment/6000.

DefaultRootWindow Macro

Purpose

Returns the root window.

Libraries

Enhanced X-Windows Library (**libX11.a**)

FORTRAN 77 Library (**libXfx.a**)

Macro Syntax

DefaultRootWindow(*DisplayPtr)*

C Syntax

Window XDefaultRootWindow(*DisplayPtr)*
**Display ****DisplayPtr*;

FORTRAN Syntax

integer*4 fxdefaultrootwindow
external fxdefaultrootwindow
integer*4 *DisplayPtr*
integer*4 *ReturnCode*
ReturnCode = **fxdefaultrootwindow(***DisplayPtr*)

Description

Both the macro and the subroutine return the root window for the default screen.

Macros can be found in the **/usr/include/X11/Xlib.h** file.

Parameter

DisplayPtr Specifies the connection to the X Server.

Error Code

BadImplementation

Implementation Specifics

This Xlib subroutine is part of AIXwindows Run Time Environment in AIXwindows Environment/6000.

DefaultScreen Macro

Purpose

Returns the default screen.

Libraries

Enhanced X-Windows Library (**libX11.a**)

FORTRAN 77 Library (**libXfx.a**)

Macro Syntax

DefaultScreen(*DisplayPtr*)

C Syntax

int XDefaultScreen(*DisplayPtr)*
Display **DisplayPtr*;

FORTRAN Syntax

integer*4 fxdefaultscreen
external fxdefaultscreen
integer*4 *DisplayPtr*
integer*4 *DefaultScreenDisplay*
DefaultScreenDisplay = **fxdefaultscreen**(*DisplayPtr*)

Description

Both the macro and the subroutine return the default screen referenced in the **XOpenDisplay** subroutine. Use this macro or subroutine to retrieve the screen number in applications that use a single screen only.

Macros can be found in the **/usr/include/X11/Xlib.h** file.

Parameter

DisplayPtr Specifies the connection to the X Server.

Error Code

BadImplementation

Implementation Specifics

This Xlib subroutine is part of AIXwindows Run Time Environment in AIXwindows Environment/6000.

DefaultScreenOfDisplay Macro

Purpose

Returns the default screen.

Libraries

Enhanced X-Windows Library (**libX11.a**)

FORTRAN 77 Library (**libXfx.a**)

Macro Syntax

DefaultScreenOfDisplay(*DisplayPtr***)**

C Syntax

Screen *XDefaultScreenOfDisplay(*DisplayPtr***)**
**Display ****DisplayPtr*;

FORTRAN Syntax

integer*4 fxdefaultscreenofdisplay
external fxdefaultscreenofdisplay
integer*4 *DisplayPtr*
integer*4 *ReturnCode*
ReturnCode = **fxdefaultscreenofdisplay(***DisplayPtr***)**

Description

Both the macro and the subroutine return a pointer to the default screen.

Macros can be found in the **/usr/include/X11/Xlib.h** file.

Parameter

DisplayPtr Specifies the connection to the X Server.

Error Code

BadImplementation

Implementation Specifics

This Xlib subroutine is part of AIXwindows Run Time Environment in AIXwindows Environment/6000.

DefaultVisual Macro

Purpose

Returns the default visual type.

Libraries

Enhanced X-Windows Library (**libX11.a**)

FORTRAN 77 Library (**libXfx.a**)

Macro Syntax

DefaultVisual(*DisplayPtr*, *ScreenNumber*)

C Syntax

Visual *XDefaultVisual(*DisplayPtr*, *ScreenNumber*)
Display **DisplayPtr*;
int *ScreenNumber*;

FORTRAN Syntax

integer*4 fxdefaultvisual
external fxdefaultvisual
integer*4 *DisplayPtr*
integer*4 *ScreenNumber*
integer*4 *DefaultVisual*
DefaultVisual = **fxdefaultvisual**(*DisplayPtr*, *ScreenNumber*)

Description

Both the macro and the subroutine return the default visual type for the specified screen.

Macros can be found in the **/usr/include/X11/Xlib.h** file.

Parameters

DisplayPtr	Specifies the connection to the X Server.
ScreenNumber	Specifies the screen number of the display device.

Error Code

BadImplementation

Implementation Specifics

This Xlib subroutine is part of AIXwindows Run Time Environment in AIXwindows Environment/6000.

DefaultVisualOfScreen Macro

Purpose

Returns the default visual.

Libraries

Enhanced X-Windows Library (**libX11.a**)

FORTRAN 77 Library (**libXfx.a**)

Macro Syntax

DefaultVisualOfScreen(*ScreenPtr*)

C Syntax

Visual *XDefaultVisualOfScreen(*ScreenPtr*)
**Screen ***ScreenPtr*;

FORTRAN Syntax

integer*4 fxdefaultvisualofscreen
external fxdefaultvisualofscreen
integer*4 *ScreenPtr*
integer*4 *ReturnCode*
ReturnCode = **fxdefaultvisualofscreen**(*ScreenPtr*)

Description

Both the macro and the subroutine return the default visual of the specified screen.

Macros can be found in the **/usr/include/X11/Xlib.h** file.

Parameter

ScreenPtr	Specifies the screen of the display device.

Error Code

BadImplementation

Implementation Specifics

This Xlib subroutine is part of AIXwindows Run Time Environment in AIXwindows Environment/6000.

DisplayCells Macro

Purpose

Returns the number of entries in the default colormap.

Libraries

Enhanced X-Windows Library (**libX11.a**)

FORTRAN 77 Library (**libXfx.a**)

Macro Syntax

DisplayCells(*DisplayPtr*,*ScreenNumber***)**

C Syntax

int XDisplayCells(*DisplayPtr*,*ScreenNumber***)**
**Display ****DisplayPtr*;
int *ScreenNumber*;

FORTRAN Syntax

integer*4 fxdisplaycells
external fxdisplaycells
integer*4 *DisplayPtr*
integer*4 *ScreenNumber*
integer*4 *ReturnCode*
ReturnCode = **fxdisplaycells(***DisplayPtr*,*ScreenNumber***)**

Description

Both the macro and the subroutine return the number of entries in the default colormap.

Macros can be found in the **/usr/include/X11/Xlib.h** file.

Parameters

DisplayPtr	Specifies the connection to the X Server.
ScreenNumber	Specifies the screen number of the display device.

Error Code

BadImplementation

Implementation Specifics

This Xlib subroutine is part of AIXwindows Run Time Environment in AIXwindows Environment/6000.

DisplayHeight Macro

Purpose

Returns an integer that describes the height of the screen in pixels.

Libraries

Enhanced X-Windows Library (**libX11.a**)

FORTRAN 77 Library (**libXfx.a**)

Macro Syntax

DisplayHeight(*DisplayPtr*, *ScreenNumber***)**

C Syntax

int XDisplayHeight(*DisplayPtr*,*ScreenNumber)*
Display ******DisplayPtr*;
int *ScreenNumber;*

FORTRAN Syntax

integer*4 fxdisplayheight
external fxdisplayheight
integer*4 *DisplayPrt*
integer*4 *ScreenNumber*
integer*4 *ReturnCode*
ReturnCode = **fxdisplayheight(***DisplayPtr*,*ScreenNumber***)**

Description

Both the macro and the subroutine return an integer that describes the height of the screen in pixels.

Macros can be found in the **/usr/include/X11/Xlib.h** file.

Parameters

DisplayPtr	Specifies the connection to the X Server.
ScreenNumber	Specifies the screen number of the display device.

Error Code

BadImplementation

Implementation Specifics

This Xlib subroutine is part of AIXwindows Run Time Environment in AIXwindows Environment/6000.

DisplayHeightMM Macro

Purpose

Returns an integer that describes the height of the screen in millimeters.

Libraries

Enhanced X-Windows Library (**libX11.a**)

FORTRAN 77 Library (**libXfx.a**)

Macro Syntax

DisplayHeightMM(*DisplayPtr*,*ScreenNumber***)**

C Syntax

int XDisplayHeightMM(*DisplayPtr*,*ScreenNumber)*
Display ******DisplayPtr*;
int *ScreenNumber*;

FORTRAN Syntax

integer*4 fxdisplayheightmm
external fxdisplayheightmm
integer*4 *DisplayPtr*
integer*4 *ScreenNumber*
integer*4 *ReturnCode*
ReturnCode **= fxdisplayheightmm(***DisplayPtr*,*ScreenNumber***)**

Description

Both the macro and the subroutine return an integer that describes the height of the screen in millimeters.

Macros can be found in the **/usr/include/X11/Xlib.h** file.

Parameters

DisplayPtr	Specifies the connection to the X Server.
ScreenNumber	Specifies the screen number of the display device.

Error Code

BadImplementation

Implementation Specifics

This Xlib subroutine is part of AIXwindows Run Time Environment in AIXwindows Environment/6000.

DisplayOfScreen Macro

Purpose

Returns the display of the specified screen.

Libraries

Enhanced X-Windows Library (**libX11.a**)

FORTRAN 77 Library (**libXfx.a**)

Macro Syntax

DisplayOfScreen(*ScreenPtr***)**

C Syntax

Display *XDisplayOfScreen(*ScreenPtr***)**
Screen ***ScreenPtr;

FORTRAN Syntax

integer*4 fxdisplayofscreen
external fxdisplayofscreen
integer*4 *DisplayPtr*
integer*4 *DisplayScreen*
DisplayScreen = **fxdisplayofscreen(***ScreenPtr***)**

Description

Both the macro and the subroutine return the display of the specified screen.

Macros can be found in the **/usr/include/X11/Xlib.h** file.

Parameter

ScreenPtr Specifies the screen of the display device.

Error Code

BadImplementation

Implementation Specifics

This Xlib subroutine is part of AIXwindows Run Time Environment in AIXwindows Environment/6000.

DisplayPlanes Macro

Purpose

Returns the depth (number of planes) of the root window of the specified screen.

Libraries

Enhanced X-Windows Library (**libX11.a**)

FORTRAN 77 Library (**libXfx.a**)

Macro Syntax

DisplayPlanes(*DisplayPtr*, *ScreenNumber***)**

C Syntax

int XDisplayPlanes(*DisplayPtr*,*ScreenNumber***)**
**Display ****DisplayPtr*;
int *ScreenNumber*;

FORTRAN Syntax

integer*4 fxdisplayplanes
external fxdisplayplanes
integer*4 *DisplayPtr*
integer*4 *ScreenNumber*
integer*4 *ReturnCode*
ReturnCode = **fxdisplayplanes(***DisplayPtr*,*ScreenNumber***)**

Description

Both the macro and the subroutine return the depth (number of planes) of the root window of the specified screen.

Macros can be found in the **/usr/include/X11/Xlib.h** file.

Parameters

DisplayPtr	Specifies the connection to the X Server.
ScreenNumber	Specifies the screen number of the display device.

Error Code

BadImplementation

Implementation Specifics

This Xlib subroutine is part of AIXwindows Run Time Environment in AIXwindows Environment/6000.

DisplayString Macro

Purpose

Obtains the string passed to the **XOpenDisplay** subroutine.

Libraries

Enhanced X-Windows Library (**libX11.a**)

FORTRAN 77 Library (**libXfx.a**)

Macro Syntax

DisplayString(*DisplayPtr***)**

C Syntax

char *XDisplayString(*DisplayPtr***)**
Display ***DisplayPtr;

FORTRAN Syntax

character*256 fxdisplaystring
external fxdisplaystring
integer*4 *DisplayPtr*
character*256 *DisplayString*
DisplayString = **fxdisplaystring(***DisplayPtr***)**

Description

Both the macro and the subroutine obtain the string passed to the **XOpenDisplay** subroutine when the current display device was opened. If the passed string is the value of **NULL**, both return the value of the **DISPLAY** environment variable when the current display was opened. This can be useful to applications that run the **fork** subroutine and have to open a new connection to the same display from the child process, as well as for printing error messages.

Macros can be found in the **/usr/include/X11/Xlib.h** file.

Parameter

DisplayPtr Specifies the connection to the X Server.

Error Code

BadImplementation

Implementation Specifics

This Xlib subroutine is part of AIXwindows Run Time Environment in AIXwindows Environment/6000.

DisplayWidth Macro

Purpose

Returns an integer that describes the width of the screen in pixels.

Libraries

Enhanced X-Windows Library (**libX11.a**)

FORTRAN 77 Library (**libXfx.a**)

Macro Syntax

DisplayWidth(*DisplayPtr,ScreenNumber*)

C Syntax

int XDisplayWidth(*DisplayPtr,ScreenNumber*)
Display **DisplayPtr*;
int *ScreenNumber*;

FORTRAN Syntax

integer*4 fxdisplaywidth
external fxdisplaywidth
integer*4 *DisplayPtr*
integer*4 *ScreenNumber*
integer*4 *ReturnCode*
ReturnCode = **fxdisplaywidth**(*DisplayPtr,ScreenNumber*)

Description

Both the macro and the subroutine return an integer that describes the width of the screen in pixels.

Macros can be found in the **/usr/include/X11/Xlib.h** file.

Parameters

DisplayPtr	Specifies the connection to the X Server.
ScreenNumber	Specifies the screen number of the display device.

Error Code

BadImplementation

Implementation Specifics

This Xlib subroutine is part of AIXwindows Run Time Environment in AIXwindows Environment/6000.

DisplayWidthMM Macro

Purpose

Returns an integer that describes the width of the screen in millimeters.

Libraries

Enhanced X-Windows Library (**libX11.a**)

FORTRAN 77 Library (**libXfx.a**)

Macro Syntax

DisplayWidthMM(*DisplayPtr*, *ScreenNumber***)**

C Syntax

int XDisplayWidthMM(*DisplayPtr*,*ScreenNumber***)**
**Display ****DisplayPtr*;
int *ScreenNumber*;

FORTRAN Syntax

integer*4 fxdisplaywidthmm
external fxdisplaywidthmm
integer*4 *DisplayPtr*
integer*4 *ScreenNumber*
integer*4 *ReturnCode*
ReturnCode **= fxdisplaywidthmm(***DisplayPtr*,*ScreenNumber***)**

Description

Both the macro and the subroutine return an integer that describes the width of the screen in millimeters.

Macros can be found in the **/usr/include/X11/Xlib.h** file.

Parameters

DisplayPtr	Specifies the connection to the X Server.
ScreenNumber	Specifies the screen number of the display device.

Error Code

BadImplementation

Implementation Specifics

This Xlib subroutine is part of AIXwindows Run Time Environment in AIXwindows Environment/6000.

DoesBackingStore Macro

Purpose

Indicates if the screen supports backing store attributes.

Libraries

Enhanced X-Windows Library (**libX11.a**)

FORTRAN 77 Library (**libXfx.a**)

Macro Syntax

DoesBackingStore(*ScreenPtr***)**

C Syntax

int XDoesBackingStore(*ScreenPtr***)**
Screen ****ScreenPtr*;**

FORTRAN Syntax

integer*4 fxdoesbackingstore
external fxdoesbackingstore
integer*4 *ScreenPtr*
integer*4 *ReturnCode*
ReturnCode **= fxdoesbackingstore(***ScreenPtr***)**

Description

Both the macro and the subroutine return a value indicating if the screen supports backing stores.

Macros can be found in the **/usr/include/X11/Xlib.h** file.

Parameter

ScreenPtr	Specifies the screen of the display device.

Return Values

Always	Maintaining contents even when the window is unmapped is beneficial.
NotUseful	Maintaining contents is unnecessary.
WhenMapped	Maintaining contents of obscured regions when the window is mapped is beneficial.

Error Code

BadImplementation

Implementation Specifics

This Xlib subroutine is part of AIXwindows Run Time Environment in AIXwindows Environment/6000.

DoesSaveUnders Macro

Purpose

Indicates if the specified screen supports the save under flag.

Libraries

Enhanced X-Windows Library (**libX11.a**)

FORTRAN 77 Library (**libXfx.a**)

Macro Syntax

DoesSaveUnders(*ScreenPtr*)

C Syntax

Bool XDoesSaveUnders(*ScreenPtr*)
Screen **ScreenPtr*;

FORTRAN Syntax

integer*4 fxdoessaveunders
external fxdoessaveunders
integer*4 *ScreenPtr*
integer*4 *ReturnCode*
ReturnCode = **fxdoessaveunders**(*ScreenPtr*)

Description

Both the macro and the subroutine return a Boolean value indicating if the specified screen supports save unders.

Macros can be found in the **/usr/include/X11/Xlib.h** file.

Parameter

ScreenPtr Specifies the screen of the display device.

Return Values

False Indicates that the screen does not support save unders.

True Indicates that the screen supports the save_unders member.

Error Code

BadImplementation

Implementation Specifics

This Xlib subroutine is part of AIXwindows Run Time Environment in AIXwindows Environment/6000.

Related Information

The **XSetWindowAttributes**, **XWindowAttributes** structures.

EventMaskOfScreen Macro

Purpose

Returns the initial evernt mask of the root window.

Libraries

Enhanced X-Windows Library (**libX11.a**)

FORTRAN 77 Library (**libXfx.a**)

Macro Syntax

EventMaskOfScreen(*ScreenPtr*)

C Syntax

long XEventMaskOfScreen(*ScreenPtr*)
Screen **ScreenPtr*;

FORTRAN Syntax

integer*4 fxeventmaskofscreen
external fxeventmaskofscreen
integer*4 *ScreenPtr*
integer*4 *ReturnCode*
ReturnCode = **fxeventmaskofscreen**(*ScreenPtr*)

Description

Both the macro and the subroutine return the initial mask of the root window for the specified screen.

Macros can be found in the **/usr/include/X11/Xlib.h** file.

Parameter

ScreenPtr Specifies the screen of the display device.

Error Code

BadImplementation

Implementation Specifics

This Xlib subroutine is part of AIXwindows Run Time Environment in AIXwindows Environment/6000.

HeightMMOfScreen Macro

Purpose

Returns an integer that describes the height of the screen in millimeters.

Libraries

Enhanced X-Windows Library (**libX11.a**)

FORTRAN 77 Library (**libXfx.a**)

Macro Syntax

HeightMMOfScreen(*ScreenPtr***)**

C Syntax

int XHeightMMOfScreen(*ScreenPtr***)**
**Screen ***ScreenPtr;*

FORTRAN Syntax

integer*4 fxheightmmofscreen
external fxheightmmofscreen
integer*4 *ScreenPtr*
integer*4 *ReturnCode*
ReturnCode = **fxheightmmofscreen(***ScreenPtr***)**

Description

Both the macro and the subroutine return an integer that describes the height of the specified screen in millimeters.

Macros can be found in the **/usr/include/X11/Xlib.h** file.

Parameter

ScreenPtr Specifies the screen of the display device.

Error Code

BadImplementation

Implementation Specifics

This Xlib subroutine is part of AIXwindows Run Time Environment in AIXwindows Environment/6000.

HeightOfScreen Macro

Purpose

Returns an integer that describes the height of the screen in pixels.

Libraries

Enhanced X-Windows Library (**libX11.a**)

FORTRAN 77 Library (**libXfx.a**)

Macro Syntax

HeightOfScreen(*ScreenPtr***)**

C Syntax

int XHeightOfScreen(*ScreenPtr***)**
Screen ****ScreenPtr*;**

FORTRAN Syntax

integer*4 fxheightofscreen
external fxheightofscreen
integer*4 *ScreenPtr*
integer*4 *ReturnCode*
ReturnCode **= fxheightofscreen(***ScreenPtr***)**

Description

Both the macro and the subroutine return an integer that describes the height of the specified screen in pixels.

Macros can be found in the **/usr/include/X11/Xlib.h** file.

Parameter

ScreenPtr Specifies the screen of the display device.

Error Code

BadImplementation

Implementation Specifics

This Xlib subroutine is part of AIXwindows Run Time Environment in AIXwindows Environment/6000.

ImageByteOrder Macro

Purpose

Specifies the required byte order.

Libraries

Enhanced X-Windows Library (**libX11.a**)

FORTRAN 77 Library (**libXfx.a**)

Macro Syntax

ImageByteOrder(*DisplayPtr***)**

C Syntax

int XImageByteOrder(*DisplayPtr***)**
Display ****DisplayPtr*;**

FORTRAN Syntax

integer*4 fximagebyteorder
external fximagebyteorder
integer*4 *DisplayPtr*
integer*4 *ReturnCode*
ReturnCode **= fximagebyteorder(***DisplayPtr***)**

Description

Both the macro and the subroutine specify the required byte order for images for each scanline unit in XY format (bitmap) or for each pixel value in Z format.

Macros can be found in the **/usr/include/X11/Xlib.h** file.

Parameter

DisplayPtr Specifies the connection to the X Server.

Return Values

LSBFirst Indicates least significant byte first.

MSBFirst Indicates most significant byte first.

Error Code

BadImplementation

Implementation Specifics

This Xlib subroutine is part of AIXwindows Run Time Environment in AIXwindows Environment/6000.

IsCursorKey Macro

Purpose

Determines if the key symbol is a cursor key.

Libraries

Enhanced X-Windows Library (**libX11.a**)

Macro Syntax

IsCursorKey(*Keysym*)

Description

The **IsCursorKey** macro returns a value of **True** if the key symbol is a cursor key.

Macros can be found in the **/usr/include/X11/Xlib.h** file.

Parameter

Keysym	Specifies the encoded symbol on a keycap of the keyboard.

Return Values

False	Indicates that the key symbol is not a cursor key.
True	Indicates that the key symbol is a cursor key.

Error Code

BadImplementation

Implementation Specifics

This Xlib subroutine is part of AIXwindows Run Time Environment in AIXwindows Environment/6000.

IsFunctionKey Macro

Purpose

Determines if the key symbol is a function key.

Libraries

Enhanced X-Windows Library (**libX11.a**)

Macro Syntax

IsFunctionKey(*Keysym***)**

Description

The **IsFunctionKey** macro returns a value of **True** if the key symbol is a function key.

Macros can be found in the **/usr/include/X11/Xlib.h** file.

Parameter

Keysym	Specifies the encoded symbol on a keycap of the keyboard.

Return Values

False	Indicates that the key symbol is not a function key.
True	Indicates that the key symbol is a function key.

Error Code

BadImplementation

Implementation Specifics

This Xlib subroutine is part of AIXwindows Run Time Environment in AIXwindows Environment/6000.

IsKeypadKey Macro

Purpose

Determines if a key symbol is a keypad key.

Libraries

Enhanced X-Windows Library (**libX11.a**)

Syntax

IsKeypadKey(*Keysym***)**

Description

The **IsKeypadKey** macro returns a value of **True** if the key symbol is a keypad key.

Macros can be found in the **/usr/includes/X11/Xlib.h** file.

Parameter

Keysym	Specifies the encoded symbol on a key cap of the keyboard.

Return Values

False	Indicates that the key symbol is not a keypad key.
True	Indicates that the key symbol is a keypad key.

Error Code

BadImplementation

Implementation Specifics

This Xlib subroutine is part of AIXwindows Run Time Environment in AIXwindows Environment/6000.

IsMiscFunctionKey Macro

Purpose

Determines if the key symbol is a miscellaneous function key.

Libraries

Enhanced X-Windows Library (**libX11.a**)

Syntax

IsMiscFunctionKey(*Keysym*)

Description

The **IsMiscFunctionKey** macro returns a value of **True** if the key symbol is a miscellaneous function key.

Macros can be found in the **/usr/include/X11/Xlib.h** file.

Parameter

Keysym Specifies the encoded symbol on a keycap of the keyboard.

Return Values

False Indicates that the key symbol is not a miscellaneous function key.

True Indicates that the key symbol is a miscellaneous function key.

Error Code

BadImplementation

Implementation Specifics

This Xlib subroutine is part of AIXwindows Run Time Environment in AIXwindows Environment/6000.

IsModifierKey Macro

Purpose

Determines if the key symbol is a modifier key.

Libraries

Enhanced X-Windows Library (**libX11.a**)

Syntax

IsModifierKey(*Keysym*)

Description

The **IsModifierKey** macro returns a value of **True** if the key symbol is a modifier key.

Macros can be found in the **/usr/include/X11/Xlib.h** file.

Parameter

Keysym Specifies the encoded symbol on a keycap of the keyboard.

Return Values

False Indicates that the key symbol is not a modifier key.

True Indicates that the key symbol is a modifier key.

Error Code

BadImplementation

Implementation Specifics

This Xlib subroutine is part of AIXwindows Run Time Environment in AIXwindows Environment/6000.

IsPFKey Macro

Purpose

Determines if the key symbol is a PF key.

Libraries

Enhanced X-Windows Library (**libX11.a**)

Syntax

IsPFKey(*Keysym*)

Description

The **IsPFKey** macro returns a value of **True** if the key symbol is a PF key.

This macro can be found in the **/usr/include/X11/Xutil.h** file.

Parameter

Keysym	Specifies the encoded symbol on a keycap of the keyboard.

Return Values

False	Indicates that the key symbol is not a PF key.
True	Indicates that the key symbol is a PF key.

Error Code

BadImplementation

Implementation Specifics

This macro is part of AIXwindows Development Environment in AIXwindows Environment/6000.

LastKnownRequestProcessed Macro

Purpose

Extracts the full serial number of the last request known by Enhanced X-Windows to have been processed by the X Server.

Libraries

Enhanced X-Windows Library (**libX11.a**)

FORTRAN 77 Library (**libXfx.a**)

Macro Syntax

LastKnownRequestProcessed(*DisplayPtr***)**

C Syntax

int XLastKnownRequestProcessed(*DisplayPtr***)**
Display ****DisplayPtr*;**

FORTRAN Syntax

integer*4 fxlastknownrequestprocessed
external fxlastknownrequestprocessed
integer*4 *DisplayPrt*
integer*4 *ReturnCode*
ReturnCode = **fxlastknownrequestprocessed(***DisplayPrt***)**

Description

Both the macro and the subroutine extract the full serial number of the last request known by Enhanced X-Windows to have been processed by the X Server. This number is automatically set when replies, events, and errors are received.

Macros can be found in the **/usr/include/X11/Xlib.h** file.

Parameter

DisplayPtr Specifies the connection to the X Server.

Error Code

BadImplementation

Implementation Specifics

This Xlib subroutine is part of AIXwindows Run Time Environment in AIXwindows Environment/6000.

MaxCmapsOfScreen Macro

Purpose

Returns the maximum number of colormaps supported by the specified screen.

Libraries

Enhanced X-Windows Library (**libX11.a**)

FORTRAN 77 Library (**libXfx.a**)

Macro Syntax

MaxCmapsOfScreen(*ScreenPtr***)**

C Syntax

int XMaxCmapsOfScreen(*ScreenPtr***)**
Screen ****ScreenPtr*;**

FORTRAN Syntax

integer*4 fxmaxcmapsofscreen
external fxmaxcmapsofscreen
integer*4 *ScreenPrt*
integer*4 *ReturnCode*
ReturnCode **= fxmaxcmapsofscreen(***ScreenPrt***)**

Description

Both the macro and the subroutine return the maximum number of colormaps supported by the specified screen.

Macros can be found in the **/usr/include/X11/Xlib.h** file.

Parameter

ScreenPtr Specifies the screen of the display device.

Error Code

BadImplementation

Implementation Specifics

This Xlib subroutine is part of AIXwindows Run Time Environment in AIXwindows Environment/6000.

MinCmapsOfScreen Macro

Purpose

Returns the minimum number of colormaps supported by the specified screen.

Libraries

Enhanced X-Windows Library (**libX11.a**)

FORTRAN 77 Library (**libXfx.a**)

Macro Syntax

MinCmapsOfScreen(*ScreenPtr***)**

C Syntax

int XMinCmapsOfScreen(*ScreenPtr***)**
**Screen ****ScreenPtr*;

FORTRAN Syntax

integer*4 fxmincmapsofscreen
external fxmincmapsofscreen
integer*4 *ScreenPtr*
integer*4 *ReturnCode*
ReturnCode = **fxmincmapsofscreen(***ScreenPtr***)**

Description

Both the macro and the subroutine return the minimum number of colormaps supported by the specified screen.

Macros can be found in the **/usr/include/X11/Xlib.h** file.

Parameter

ScreenPtr Specifies the screen of the display device.

Error Code

BadImplementation

Implementation Specifics

This Xlib subroutine is part of AIXwindows Run Time Environment in AIXwindows Environment/6000.

NextRequest Macro

Purpose

Extracts the full serial number to be used for the next request.

Libraries

Enhanced X-Windows Library (**libX11.a**)

FORTRAN 77 Library (**libXfx.a**)

Macro Syntax

NextRequest(*DisplayPtr***)**

C Syntax

int XNextRequest(*DisplayPtr***)**
**Display ****DisplayPtr;*

FORTRAN Syntax

integer*4 fxnextrequest
external fxnextrequest
integer*4 *DisplayPtr*
integer*4 *ReturnCode*
ReturnCode **= fxnextrequest(***DisplayPtr***)**

Description

Both the macro and the subroutine extract the full serial number to be used for the next request. Serial numbers are maintained separately for each display connection.

Macros can be found in the **/usr/include/X11/Xlib.h** file.

Parameter

DisplayPtr Specifies the connection to the X Server.

Error Code

BadImplementation

Implementation Specifics

This Xlib subroutine is part of AIXwindows Run Time Environment in AIXwindows Environment/6000.

PlanesOfScreen Macro

Purpose

Returns the number of planes (depth) in the specified screen.

Libraries

Enhanced X-Windows Library (**libX11.a**)

FORTRAN 77 Library (**libXfx.a**)

Macro Syntax

PlanesOfScreen(*ScreenPtr***)**

C Syntax

int XPlanesOfScreen(*ScreenPtr***)**
Screen ****ScreenPtr*;**

FORTRAN Syntax

integer*4 fxplanesofscreen
external fxplanesofscreen
integer*4 *ScreenPtr*
integer*4 *ReturnCode*
ReturnCode = **fxplanesofscreen(***ScreenPtr***)**

Description

Both the macro and the subroutine return the number of planes (depth) in the specified screen.

Macros can be found in the **/usr/include/X11/Xlib.h** file.

Parameter

ScreenPtr Specifies the screen of the display device.

Error Code

BadImplementation

Implementation Specifics

This Xlib subroutine is part of AIXwindows Run Time Environment in AIXwindows Environment/6000.

ProtocolRevision Macro

Purpose

Returns the minor protocol revision number.

Libraries

Enhanced X-Windows Library (**libX11.a**)

FORTRAN 77 Library (**libXfx.a**)

Macro Syntax

ProtocolRevision(*DisplayPtr***)**

C Syntax

int XProtocolRevision(*DisplayPtr***)**
Display ****DisplayPtr*;**

FORTRAN Syntax

integer*4 fxprotocolrevision
external fxprotocolrevision
integer*4 *DisplayPtr*
integer*4 *ReturnCode*
ReturnCode **= fxprotocolrevision(***DisplayPtr***)**

Description

Both the macro and the subroutine return the minor protocol revision number (0) of the X Server associated with the display device.

Macros can be found in the **/usr/include/X11/Xlib.h** file.

Parameter

DisplayPtr Specifies the connection to the X Server.

Error Code

BadImplementation

Implementation Specifics

This Xlib subroutine is part of AIXwindows Run Time Environment in AIXwindows Environment/6000.

ProtocolVersion Macro

Purpose

Returns the major version number.

Libraries

Enhanced X-Windows Library (**libX11.a**)

FORTRAN 77 Library (**libXfx.a**)

Macro Syntax

ProtocolVersion(*DisplayPtr***)**

C Syntax

int XProtocolVersion(*DisplayPtr)*
**Display ****DisplayPtr;*

FORTRAN Syntax

integer*4 fxprotocolversion
external fxprotocolversion
integer*4 *DisplayPrt*
integer*4 *ReturnCode*
ReturnCode = **fxprotocolversion(***DisplayPrt***)**

Description

Both the macro and the subroutine return the major version number (11) of the Enhanced X-Windows protocol associated with the display device.

Macros can be found in the **/usr/include/X11/Xlib.h** file.

Parameter

DisplayPtr Specifies the connection to the X Server.

Error Code

BadImplementation

Implementation Specifics

This Xlib subroutine is part of AIXwindows Run Time Environment in AIXwindows Environment/6000.

QLength Macro

Purpose

Returns the length of the event queue for the display.

Libraries

Enhanced X-Windows Library (**libX11.a**)

FORTRAN 77 Library (**libXfx.a**)

Macro Syntax

QLength(*DisplayPtr***)**

C Syntax

int XQLength(*DisplayPtr)*
**Display ****DisplayPtr;*

FORTRAN Syntax

integer*4 fxqlength
external fxqlength
integer*4 *DisplayPtr*
integer*4 *ReturnCode*
ReturnCode **= fxqlength(***DisplayPtr***)**

Description

Both the macro and the subroutine return the length of the event queue for the display device. There may be other events that have not been read into the queue yet.

Macros can be found in the **/usr/include/X11/Xlib.h** file.

Parameter

DisplayPtr Specifies the connection to the X Server.

Implementation Specifics

This Xlib subroutine is part of AIXwindows Run Time Environment in AIXwindows Environment/6000.

RootWindow Macro

Purpose

Returns the root window.

Libraries

Enhanced X-Windows Library (**libX11.a**)

FORTRAN 77 Library (**libXfx.a**)

Macro Syntax

RootWindow(*DisplayPtr, ScreenNumber)*

C Syntax

Window XRootWindow(*DisplayPtr,ScreenNumber)*
Display **DisplayPtr*;
int *ScreenNumber;*

FORTRAN Syntax

integer*4 fxrootwindow
external fxrootwindow
integer*4 *DisplayPtr*
integer*4 *ScreenNumber*
integer*4 *ReturnCode*
ReturnCode = **fxrootwindow(***DisplayPtr,ScreenNumber*)

Description

Both the macro and the subroutine return the root window. This is useful with subroutines that need a drawable of a particular screen, and for creating top level windows.

Macros can be found in the **/usr/include/X11/Xlib.h** file.

Parameters

DisplayPtr	Specifies the connection to the X server.
ScreenNumber	Specifies the screen number of the display device.

Error Code

BadImplementation

Implementation Specifics

This Xlib subroutine is part of AIXwindows Run Time Environment in AIXwindows Environment/6000.

RootWindowOfScreen Macro

Purpose

Returns the root window of the specified screen.

Libraries

Enhanced X-Windows Library (**libX11.a**)

FORTRAN 77 Library (**libXfx.a**)

Macro Syntax

RootWindowOfScreen(*ScreenPtr***)**

C Syntax

Window XRootWindowOfScreen(*ScreenPtr)*
**Screen ****ScreenPtr*;

FORTRAN Syntax

integer*4 fxrootwindowofscreen
external fxrootwindowofscreen
integer*4 *ScreenPrt*
integer*4 *ReturnCode*
ReturnCode = **fxrootwindowofscreen(***ScreenPrt***)**

Description

Both the macro and the subroutine return the root window of the specified screen.

Macros can be found in the **/usr/include/X11/Xlib.h** file.

Parameter

ScreenPtr Specifies the screen of the display device.

Error Code

BadImplementation

Implementation Specifics

This Xlib subroutine is part of AIXwindows Run Time Environment in AIXwindows Environment/6000.

ScreenCount Macro

Purpose

Returns the number of available screens.

Libraries

Enhanced X-Windows Library (**libX11.a**)

FORTRAN 77 Library (**libfx.a**)

Macro Syntax

ScreenCount(*DisplayPtr***)**

C Syntax

int XScreenCount(*DisplayPtr)*
**Display ****DisplayPtr*;

FORTRAN Syntax

integer*4 fxscreencount
external fxscreencount
integer*4 *DisplayPrt*
integer*4 *ReturnCode*
ReturnCode **= fxscreencount(***DisplayPrt***)**

Description

Both the macro and the subroutine return the number of available screens.

Macros can be found in the **/usr/include/X11/Xlib.h** file.

Parameter

DisplayPtr Specifies the connection to the X Server.

Error Code

BadImplementation

Implementation Specifics

This Xlib subroutine is part of AIXwindows Run Time Environment in AIXwindows Environment/6000.

ScreenOfDisplay Macro

Purpose

Returns a pointer to the screen of the specified display.

Libraries

Enhanced X-Windows Library (**libX11.a**)

FORTRAN 77 Library (**libXfx.a**)

Macro Syntax

ScreenOfDisplay(*DisplayPtr, ScreenNumber)*

C Syntax

Screen *XScreenOfDisplay(*DisplayPtr,ScreenNumber)*
**Display ***DisplayPtr*;
int *ScreenNumber;*

FORTRAN Syntax

integer*4 fxscreenofdisplay
external fxscreenofdisplay
integer*4 *DisplayPtr*
integer*4 *ScreenNumber*
integer*4 *ScreenDisplay*
ScreenDisplay = **fxscreenofdisplay(***DisplayPtr,ScreenNumber)*

Description

Both the macro and the subroutine return a pointer to the screen of the specified display.

Macros can be found in the **/usr/include/X11/Xlib.h** file.

Parameters

DisplayPtr	Specifies the connection to the X Server.
ScreenNumber	Specifies the screen number of the display.

Error Code

BadImplementation

Implementation Specifics

This Xlib subroutine is part of AIXwindows Run Time Environment in AIXwindows Environment/6000.

ServerVendor Macro

Purpose

Returns a pointer to a null–terminated string.

Libraries

Enhanced X-Windows Library (**libX11.a**)

FORTRAN 77 Library (**libfx.a**)

Macro Syntax

ServerVendor(*DisplayPtr***)**

C Syntax

char *XServerVendor(*DisplayPtr***)**
Display ***DisplayPtr;

FORTRAN Syntax

character*256 fxservervendor
external fxservervendor
integer*4 *DisplayPtr*
character*256 *ServerVendor*
ServerVendor = **fxservervendor(***DisplayPtr***)**

Description

Both the macro and the subroutine return a pointer to a null–terminated string that provides some identification of the owner of the X Server implementation.

Macros can be found in the **/usr/include/X11/Xlib.h** file.

Parameter

DisplayPtr Specifies the connection to the X Server.

Error Code

BadImplementation

Implementation Specifics

This Xlib subroutine is part of AIXwindows Run Time Environment in AIXwindows Environment/6000.

VendorRelease Macro

Purpose

Returns a number related to a vendor's release of the X Server.

Libraries

Enhanced X-Windows Library (**libX11.a**)

FORTRAN 77 Library (**libXfx.a**)

Macro Syntax

VendorRelease(*DisplayPtr***)**

C Syntax

int XVendorRelease(*DisplayPtr)*
**Display ****DisplayPtr;*

FORTRAN Syntax

integer*4 fxvendorrelease
external fxvendorrelease
integer*4 *DisplayPrt*
integer*4 *ReturnCode*
ReturnCode **= fxvendorrelease(***DisplayPtr***)**

Description

Both the macro and the subroutine return a number related to a vendor's release of the X Server.

Macros can be found in the **/usr/include/X11/Xlib.h** file.

Parameter

DisplayPtr Specifies the connection to the X Server.

Error Code

BadImplementation

Implementation Specifics

This Xlib subroutine is part of AIXwindows Run Time Environment in AIXwindows Environment/6000.

WhitePixel Macro

Purpose

Returns the white pixel value for the specified screen.

Libraries

Enhanced X-Windows Library (**libX11.a**)

FORTRAN 77 Library (**libXfx.a**)

Macro Syntax

WhitePixel(*DisplayPtr*, *ScreenNumber*)

C Syntax

unsigned long XWhitePixel(*DisplayPtr*,*ScreenNumber*)
Display **DisplayPtr*;
int *ScreenNumber*;

FORTRAN Syntax

integer*4 fxwhitepixel
external fxwhitepixel
integer*4 *DisplayPrt*
integer*4 *ScreenNumber*
integer*4 *ReturnCode*
ReturnCode = **fxwhitepixel**(*DisplayPrt*,*ScreenNumber*)

Description

Both the macro and the subroutine return the white pixel value for the specified screen.

Macros can be found in the **/usr/include/X11/Xlib.h** file.

Parameters

DisplayPtr	Specifies the connection to the X Server.
ScreenNumber	Specifies the screen number of the display device.

Error Code

BadImplementation

Implementation Specifics

This Xlib subroutine is part of AIXwindows Run Time Environment in AIXwindows Environment/6000.

WhitePixelOfScreen Macro

Purpose

Returns the white pixel value.

Libraries

Enhanced X-Windows Library (**libX11.a**)

FORTRAN 77 Library (**libXfx.a**)

Macro Syntax

WhitePixelOfScreen(*ScreenPtr***)**

C Syntax

unsigned long XWhitePixelOfScreen(*ScreenPtr***)**
**Screen ****ScreenPtr*;

FORTRAN Syntax

integer*4 fxwhitepixelofscreen
external fxwhitepixelofscreen
integer*4 *ScreenPtr*
integer*4 *ReturnCode*
ReturnCode = **fxwhitepixelofscreen(***ScreenPtr***)**

Description

Both the macro and the subroutine return the white pixel value of the specified screen.

Macros can be found in the **/usr/include/X11/Xlib.h** file.

Parameter

ScreenPtr Specifies the screen of the display device.

Error Code

BadImplementation

Implementation Specifics

This Xlib subroutine is part of AIXwindows Run Time Environment in AIXwindows Environment/6000.

WidthMMOfScreen Macro

Purpose

Returns an integer that describes the width of the screen in millimeters.

Libraries

Enahced X-Windows Library (**libX11.a**)

FORTRAN 77 Library (**libXfx.a**)

Macro Syntax

WidthMMOfScreen(*ScreenPtr***)**

C Syntax

int XWidthMMOfScreen(*ScreenPtr)*
Screen ******ScreenPtr*;

FORTRAN Syntax

integer*4 fxwidthmmofscreen
external fxwidthmmofscreen
integer*4 *ScreenPtr*
integer*4 *ReturnCode*
ReturnCode = **fxwidthmmofscreen(***ScreenPtr***)**

Description

Both the macro and the subroutine return an integer that describes the width of the specified screen in millimeters.

Macros can be found in the **/usr/include/X11/Xlib.h** file.

Parameter

ScreenPtr Specifies the screen of the display device.

Error Code

BadImplementation

Implementation Specifics

This Xlib subroutine is part of AIXwindows Run Time Environment in AIXwindows Environment/6000.

WidthOfScreen Macro

Purpose

Returns an integer that describes the width of the screen in pixels.

Libraries

Enhanced X-Windows Library (**libX11.a**)

FORTRAN 77 Library (**libXfx.a**)

Macro Syntax

WidthOfScreen(*ScreenPtr)*

C Syntax

int XWidthOfScreen(*ScreenPtr)*
**Screen ****ScreenPtr*;

FORTRAN Syntax

integer*4 fxwidthofscreen
external fxwidthofscreen
integer*4 *ScreenPrt*
integer*4 *ReturnCode*
ReturnCode = **fxwidthofscreen(***ScreenPrt***)**

Description

Both the macro and the subroutine return an integer that describes the width of the specified screen in pixels.

Macros can be found in the **/usr/include/X11/Xlib.h** file.

Parameter

ScreenPtr Specifies the screen of the display device.

Error Code

BadImplementation

Implementation Specifics

This Xlib subroutine is part of AIXwindows Run Time Environment in AIXwindows Environment/6000.

XActivateScreenSaver Subroutine

Purpose

Activates the screen saver.

Libraries

Enhanced X-Windows Library (**libX11.a**)

FORTRAN 77 Library (**libXfx.a**)

C Syntax

XActivateScreenSaver(*DisplayPtr***)**
**Display ****DisplayPtr;*

FORTRAN Syntax

external fxactivatescreensaver
integer*4 *DisplayPtr*
call fxactivatescreensaver(*DisplayPtr***)**

Description

The **XActivateScreenSaver** subroutine activates the screen saver.

Parameter

DisplayPtr Specifies the connection to the X Server.

Error Code

BadImplementation

Implementation Specifics

This Xlib subroutine is part of AIXwindows Run Time Environment in AIXwindows Environment/6000.

Related Information

The **XSetScreenSaver** subroutine.

The **ForceScreenSaver** protocol request.

XAddHost Subroutine

Purpose

Allows access for the specified host to the display device.

Libraries

Enhanced X-Windows Library (**libX11.a**)

FORTRAN 77 Library (**libXfx.a**)

C Syntax

XAddHost(*DisplayPtr*, *Host***)**
Display ****DisplayPtr;***
XHostAddress ****Host;***

FORTRAN Syntax

external fxaddhost
integer*4 *DisplayPrt,Host*
call fxaddhost(*DisplayPtr,Host***)**

Description

The **XAddHost** subroutine adds the specified host to the access control list for that display device. The display device (server) and the program (client) must be on the same host.

Parameters

DisplayPtr	Specifies the connection to the X Server.
Host	Specifies the network address of the host machine.

Error Codes

BadAccess

BadImplementation

BadValue

Implementation Specifics

This Xlib subroutine is part of AIXwindows Run Time Environment in AIXwindows Environment/6000.

Related Information

The **ChangeHosts** protocol request.

XAddHosts Subroutine

Purpose

Allows access for multiple hosts to the specified display device.

Libraries

Enhanced X-Windows Library (**libX11.a**)

FORTRAN 77 Library (**libXfx.a**)

C Syntax

XAddHosts(*DisplayPtr*, *Hosts*, *NumberHosts***)**
Display ******DisplayPtr*;
XHostAddress ******Hosts;*
int *NumberHosts*;

FORTRAN Syntax

external fxaddhosts
integer*4 *DisplayPrt,Hosts,NumberHosts*
call fxaddhosts(*DisplayPrt,Hosts,NumberHosts***)**

Description

The **XAddHosts** subroutine adds each specified host to the access control list for that display device. The display device (server) and the program (client) must be on the same host.

Parameters

DisplayPtr	Specifies the connection to the X Server.
Hosts	Specifies each host to be added.
NumberHosts	Specifies the number of hosts to be added.

Error Codes

BadAccess

BadImplementation

BadValue

Implementation Specifics

This Xlib subroutine is part of AIXwindows Run Time Environment in AIXwindows Environment/6000.

Related Information

The **ChangeHosts** protocol request.

The **XAddHost** subroutine.

XAddPixel Subroutine

Purpose

Adds a value to every pixel in an image.

Libraries

Enhanced X-Windows Library (**libX11.a**)

FORTRAN 77 Library (**libXfx.a**)

C Syntax

int XAddPixel(*XimagePtr*, *Value***)**
XImage ****XImagePtr;***
int *Value*;

FORTRAN Syntax

integer*4 fxaddpixel
external fxaddpixel
integer*4 *XimagePtr*,*Value*
integer*4 *Status*
Status = **fxaddpixel(***XimagePtr*,*Value***)**

Description

The **XAddPixel** subroutine adds a value to every pixel in an image. Use this subroutine to manipulate the base pixel value for allocating color resources to an image.

Parameters

XimagePtr	Specifies a pointer to the image.
Value	Specifies the value to be added.

Error Code

BadImplementation

Implementation Specifics

This Xlib subroutine is part of AIXwindows Run Time Environment in AIXwindows Environment/6000.

XAddToSaveSet Subroutine

Purpose

Adds a window to the client's saveset.

Libraries

Enhanced X-Windows Library (**libX11.a**)

FORTRAN 77 Library (**libXfx.a**)

C Syntax

XAddToSaveSet(*DisplayPtr*, *WindowAdd***)**
Display **DisplayPtr*;
Window *WindowAdd*;

FORTRAN Syntax

external fxaddtosaveset
integer*4 *DisplayPtr*
integer*4 *WindowAdd*
call fxaddtosaveset(*DisplayPtr*, *WindowAdd***)**

Description

The **XAddToSaveSet** subroutine adds the window and the children of the window specified to the client saveset. The specified window must be created by another client. The X Server automatically removes the windows from the saveset when the specified window is destroyed.

Parameters

DisplayPtr	Specifies the connection to the X Server.
WindowAdd	Specifies the window ID of the window to be added.

Error Codes

BadImplementation

BadMatch

BadWindow

Implementation Specifics

This Xlib subroutine is part of AIXwindows Run Time Environment in AIXwindows Environment/6000.

Related Information

The **ChangeSaveSet** protocol request.

XAllocColor Subroutine

Purpose

Allocates a read–only color cell.

Libraries

Enhanced X-Windows Library (**libX11.a**)

FORTRAN 77 Library (**libXfx.a**)

C Syntax

Status XAllocColor(*DisplayPtr, ColormapID, ColorValues***)**
Display ******DisplayPtr;*
Colormap *ColormapID;*
XColor ******ColorValues;*

FORTRAN Syntax

integer*4 fxalloccolor
external fxalloccolor
integer*4 *DisplayPtr*
integer*4 *ColormapID*
integer*4 *ColorValues*
integer*4 *Status*
Status = **fxalloccolor(***DisplayPtr,ColormapID,ColorValues***)**

Description

The **XAllocColor** subroutine 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. Read-only colormap cells are shared among clients. When the last client deallocates a shared cell, the colormap cell is deallocated.

Parameters

DisplayPtr	Specifies the connection to tne X Server.
ColormapID	Specifies the colormap ID.
ColorValues	Specifies the RGB values that it wants to use, and returns the pixel value and RGB values in the colormap.

Return Values

0	Indicates that is unsuccessful.
Non-zero	Indicates that is successful.

Error Codes

BadColor

BadImplementation

Implementation Specifics

This Xlib subroutine is part of AIXwindows Run Time Environment in AIXwindows Environment/6000.

Related Information

The **AllocColor** protocol request.

XAllocColorCells Subroutine

Purpose

Allocates read-write color cells.

Libraries

Enhanced X-Windows Library (**libX11.a**)

FORTRAN 77 Library (**libXfx.a**)

C Syntax

Status XAllocColorCells(*DisplayPtr, ColormapID, Contiguous,*
PlaneMaskReturn, NumberPlanes,
PixelsReturn, NumberPixels)
Display **DisplayPtr;*
Colormap *ColormapID;*
Bool *Contiguous;*
unsigned long *PlaneMaskReturn[ ];*
unsigned int *NumberPlanes;*
unsigned long *PixelsReturn[ ];*
unsigned int *NumberPixels;*

FORTRAN Syntax

integer*4 fxalloccolorcells
external fxalloccolorcells
integer*4 *DisplayPtr*
integer*4 *ColormapID*
integer*4 *Contiguous,PlaneMasksReturn*
integer*4 *NumberPlanes,PixelsReturn,NumberPixels*
integer*4 *Status*
Status = **fxalloccolorcells(***DisplayPtr,ColormapID,Contiguous,*
PlaneMaskReturn,NumberPlanes,
PixelsReturn,NumberPixels)

Description

The **XAllocColorCells** subroutine allocates color cells. The number of colors must be positive and the number of planes must be nonnegative.

If the planes must be contiguous, set the *Contiguous* parameter to a value of 1. If the planes do not need to be contiguous, set the *Contiguous* parameter to a value of 0.

If *NumberPlanes* and *NumberPixels* are requested, *NumberPlanes* plane masks and *NumberPixels* 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, NumberPixels* 2**NumberPlanes distinct pixel values can be produced. These pixel values are allocated writable by the request.

If *the Contiguous* parameter has a value of **True** and all masks are combined, the following occur:

- A single contiguous set of bits is formed for the **GrayScale** or **PseudoColor** visual type. Each mask has one bit.

- Three contiguous sets of bits (one within each pixel subfield) are formed for the **DirectColor** visual type. Each mask has 3 bits.

The RGB values of the allocated entries are not defined.

Parameters

ColormapID	Specifies the colormap ID.
Contiguous	Specifies a Boolean value.
DisplayPtr	Specifies the connection to the X Server.
NumberPixels	Specifies the number of pixel values returned in the *PixelsReturn* parameter.
NumberPlanes	Specifies the number of plane masks returned in the *PlaneMaskReturn*.
PixelsReturn	Returns a list of pixel values.
PlaneMaskReturn	Returns a list of plane masks.

Return Values

Nonzero	Indicates that the **XAllocColorCells** subroutine succeeds.
0	Indicates that the **XAllocColorCells** subroutine does not succeed.

Error Codes

BadAlloc

BadImplementation

BadValue

Implementation Specifics

This Xlib subroutine is part of AIXwindows Run Time Environment in AIXwindows Environment/6000.

Related Information

The **AllocColorCells** protocol request.

XAllocColorPlanes Subroutine

Purpose

Allocates color planes.

Libraries

Enhanced X-Windows Library (**libX11.a**)

FORTRAN 77 Library (**libXfx.a**)

C Syntax

Status XAllocColorPlanes(*DisplayPtr*, *ColormapID*,
Contiguous, *PixelsReturn*,
NumberColors, *NumberReds*,*NumberGreens*,
NumberBlues, *RedMaskReturn*,
GreenMaskReturn, *BlueMaskReturn*)
Display **DisplayPtr*;
Colormap *ColormapID*;
Bool *Contiguous*;
unsigned long *PixelsReturn*[];
int *NumberColors*;
int *NumberReds*,*NumberGreens*,*NumberBlues*;
unsigned long **RedMaskReturn*,**GreenMaskReturn*,**BlueMaskReturn*;

FORTRAN Syntax

integer*4 fxalloccolorplanes
external fxalloccolorcells
integer*4 *DisplayPtr*
integer*4 *ColormapID*
integer*4 *Contiguous*,*PixelsReturn*
integer*4 *NumberColors*,*NumberReds*,*NumberGreens*,*NumberBlues*
integer*4 *RedMaskReturn*,*GreenMaskReturn*
integer*4 *BlueMaskReturn*
integer*4 *Status*
Status = **fxalloccolorplanes**(*DisplayPtr*,*ColormapID*,*Contiguous*,
PixelsReturn,*NumberColors*,*NumberReds*,
NumberGreens,*NumberBlues*,*RedMaskReturn*,
BlueMaskReturn,*GreenMaskReturn*)

Description

The **XAllocColorPlanes** subroutine allocates color planes. It returns the pixel values in the *PixelsReturn* parameter.

If the *NumberColors* parameter colors, *NumberReds* parameter reds, *NumberGreens* parameter greens, and the *NumberBlues* parameter blues are requested, the *NumberColors* parameter pixels are returned. The masks returned have the bits for the *NumberReds*, *NumberGreens*, and *NumberBlues* parameters set respectively.

For, the **DirectColor** and **PseudoColor** visual type, each mask lies within the corresponding pixel subfield. Distinct pixels values can be produced by combining subsets of masks with pixels as follows:

NumberColors * 2*******NumberReds* + *NumberGreens* + *NumberBlues*

The combinations are allocated by the request. However, the colormap only contains the following:

NumberColors * 2**_NumberReds_	independent red entries
NumberColors * 2**_NumberGreens_	independent green entries
Numbercolors * 2**_NumberBlues_	independent blue entries

When the colormap entry for a pixel value is changed with the **XStoreColor**, the **XStoreColors** or the **XStoreNamedColor** subroutines, the pixel is decomposed according to the masks, and corresponding independent entries are updated.

Parameters

BlueMaskReturn	Returns the bit masks for the blue planes.
ColormapID	Specifies the colormap ID.
Contiguous	Specifies a Boolean value. If the value of **True**, each mask has a contiguous set of bits. No mask has any bits in common with any other mask or with any of the pixels. If the planes must be contiguous, set the *Contiguous* parameter to the value of **1**. If the planes do not need to be contiguous, set the *Contiguous* parameter to the value of **0.**
DisplayPtr	Specifies the connection to the X Server.
GreenMaskReturn	Returns the bit masks for the green planes.
NumberBlues	Specifies the number of blue color planes. This value must be a nonnegative value.
NumberColors	Specifies the number of pixel values that are to be returned in the PixelsReturn parameter.
NumberGreens	Specifies the number of green color planes. This value must be a nonnegative value.
NumberReds	Specifies the number of red color planes. This value must be a nonnegative value.
PixelsReturn	Returns an array of pixel values.
RedMaskReturn	Returns the bit masks for the red planes.

Return Values

0	Indicates that the **XAllocColorPlanes** subroutine does not succeed.
Nonzero	Indicates that the **XAllocColorPlanes** subroutine succeeds.

Error Codes

BadAlloc

BadColor

BadImplementation

BadValue

Implementation Specifics

This Xlib subroutine is part of AIXwindows Run Time Environment in AIXwindows Environment/6000.

Related Information

The **AllocColorPlanes** protocol request.

XAllocNamedColor Subroutine

Purpose

Allocates a read-only color cell by name.

Libraries

Enhanced X-Windows Library (**libX11.a**)

FORTRAN 77 Library (**libXfx.a**)

C Syntax

Status XAllocNamedColor(*DisplayPtr*, *ColormapID*, *ColorName*,
ScreenDefinitionReturn,
ExactDefinitionReturn)
Display **DisplayPtr*;
Colormap *ColormapID*;
char **ColorName*;
XColor **ScreenDefinitionReturn*,**ExactDefinitionReturn*;

FORTRAN Syntax

integer*4 fxallocnamedcolor
external fxallocnamedcolor
integer*4 *DisplayPtr*
integer*4 *ColormapID*
character*256 *ColorName*
integer*4 *ScreenDefinitionReturn*
integer*4 *ExactDefinitionReturn*
integer*4 *Status*
Status = **fxallocnamedcolor**(*DisplayPtr*,*ColormapID*,*ColorName*,
ScreenDefinitionReturn,
ExactDefinitionReturn)

Description

The **XAllocNamedColor** subroutine obtains the color definition structure for a specified color. It determines the correct color or shade for the screen. It returns the exact database definition, and the closest available color supported by the hardware. The allocated color cell is read-only.

Parameters

ColormapID	Specifies the colormap ID.
ColorName	Specifies the color name string for the color definition structure to be returned. The color name is not case-sensitive.
DisplayPtr	Specifies the connection to the X Server.
ExactDefinitionReturn	Returns the actual pixel values and closest RGB values provided by the hardware for the color name specified.
ScreenDefinitionReturn	Returns the values used in the colormap.

Return Values

0	Indicates that the **XAllocNamedColor** subroutine is unsuccessful.
Nonzero	Indicates that the **XAllocNamedColor** subroutine is successful.

Error Codes

BadAlloc

BadColor

BadImplementation

BadName

Implementation Specifics

This Xlib subroutine is part of AIXwindows Run Time Environment in AIXwindows Environment/6000.

Related Information

The **AllocNamedColor** protocol request.

XAllowEvents Subroutine

Purpose

Releases some queued events if the client has frozen a device.

Libraries

Enhanced X-Windows Library (**libX11.a**)

FORTRAN 77 Library (**libXfx.a**)

C Syntax

XAllowEvents(*DisplayPtr*, *EventMode*, *TimeStamp*)
Display **DisplayPtr*;
int *EventMode*;
Time *TimeStamp*;

FORTRAN Syntax

external fxallowevents
integer*4 *DisplayPtr*
integer*4 *EventMode*,*TimeStamp*
call fxallowevents(*DisplayPtr*,*EventMode*,*TimeStamp*)

Description

The **XAllowEvents** subroutine 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.

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.

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, the **AsyncPointer** event mode value 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.

The **SyncPointer** event mode value 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 the

XGrabButton subroutine or a previous **XAllowEvents** subroutine with mode the **SyncPointer** subroutine, but not from the **XGrabPointer** subroutine, the pointer grab is released and that event is completely reprocessed. This time, however, the **XAllowEvents** subroutine ignores any passive grabs at or above (toward the root) the *GrabWindow* parameter of the grab just released.

ReplayPointer event mode value 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, the **AsyncKeyboard** event mode value releases for both.

The **AsyncKeyboard** event mode value 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.

The **SyncKeyboard** event mode value 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 the **XGrabKey** subroutine or a previous **XAllowEvents** subroutine with mode the **SyncKeyboard** subroutine, but not from an **XGrabKeyboard** subroutine, the keyboard grab is released and that event is completely reprocessed. This time, however, the **XAllowEvents** subroutine ignores any passive grabs at or above (toward the root) the *GrabWindow* parameter of the grab just released.

The **ReplayKeyboard** event mode value 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, the devices do not freeze. If the other device is still grabbed, then a subsequent event still causes both devices to freeze.

The **SyncBoth** event mode value has no effect unless both the 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, the **SyncBoth** event mode value releases for both grabs (but subsequent holds on the **SyncBoth** event mode value freezes each device only once).

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, the **AsyncBoth** event mode value releases for both.

The **AsyncBoth** event mode value has no effect unless both pointer and keyboard are frozen by the client.

AsyncPointer, **SyncPointer**, and **ReplayPointer** event mode values have no effect on the processing of keyboard events. The **AsyncKeyboard**, **SyncKeyboard**, and **ReplayKeyboard** event mode values have no effect on the processing of pointer events.

Parameters

DisplayPtr Specifies the connection to the X Server.

EventMode Specifies the event mode, which describes the processing that occurs.

TimeStamp Specifies the timestamp, which is expressed in milliseconds, or the value of **CurrentTime**.

Error Code

BadValue

Implementation Specifics

This Xlib subroutine is part of AIXwindows Run Time Environment in AIXwindows Environment/6000.

Related Information

The **AllowEvents** protocol request.

XAutoRepeatOff Subroutine

Purpose

Turns off keyboard the auto-repeat setting.

Libraries

Enhanced X-Windows Library (**libX11.a**)

FORTRAN 77 Library (**libXfx.a**)

C Syntax

XAutoRepeatOff(*DisplayPtr***)**
**Display ****DisplayPtr*;

FORTRAN Syntax

external fxautorepeatoff
integer*4 *DisplayPtr*
call fxautorepeatoff(*DisplayPtr***)**

Description

The **XAutoRepeatOff** subroutine turns off the auto-repeat setting for the keyboard on the specified display device.

Parameter

DisplayPtr Specifies the connection to the X Server.

Implementation Specifics

This Xlib subroutine is part of AIXwindows Run Time Environment in AIXwindows Environment/6000.

Related Information

The **ChangeKeyboardControl** protocol request.

The **XAutoRepeatOn** subroutine.

XAutoRepeatOn Subroutine

Purpose

Turns on keyboard the auto-repeat setting.

Libraries

Enhanced X-Windows Library (**libX11.a**)

FORTRAN 77 Library (**libXfx.a**)

C Syntax

XAutoRepeatOn(*DisplayPtr***)**
Display ****DisplayPtr*;**

FORTRAN Syntax

external fxautorepeaton
integer*4 *DisplayPtr*
call fxautorepeaton(*DisplayPtr***)**

Description

The **XAutoRepeatOn** subroutine turns on the auto-repeat setting for the keyboard on the specified display device.

Parameter

DisplayPtr Specifies the connection to the X Server.

Implementation Specifics

This Xlib subroutine is part of AIXwindows Run Time Environment in AIXwindows Environment/6000.

Related Information

The **ChangeKeyboardControl** protocol request.

The **XAutoRepeatOff** subroutine.

XBell Subroutine

Purpose

Sets the volume of the bell.

Libraries

Enhanced X-Windows Library (**libX11.a**)

FORTRAN 77 Library (**libXfx.a**)

C Syntax

XBell(*DisplayPtr*, *Percent*)
Display **DisplayPtr*;
int *Percent*;

FORTRAN Syntax

external fxbell
integer*4 *DisplayPtr,Percent*
call fxbell(*DisplayPtr,Percent*)

Description

The **XBell** subroutine rings the bell on the keyboard of the specified display device, if possible. The specified volume is relative to the base volume for the keyboard. If the value for the *Percent* parameter is not in the range from -100 to 100%, inclusive an error is generated. The volume at which the bell is rung when the *Percent* parameter is nonnegative is the following:

```
base – [(base * percent) / 100] + percent
```

The volume at which the bell is rung when the *Percent* parameter is negative is the following:

```
base + [(base * percent) / 100]
```

To change the base volume of the bell for this keyboard, use the **XChangeKeyboardControl** subroutine.

Parameters

DisplayPtr	Specifies the connection to the X Server.
Percent	Specifies the base volume for the bell. The volume can range from -100 to 100% inclusive.

Error Codes

BadImplementation

BadValue

Implementation Specifics

This Xlib subroutine is part of AIXwindows Run Time Environment in AIXwindows Environment/6000.

Related Information

The **Bell** protocol request.

XChangeActivePointerGrab Subroutine

Purpose

Changes the active pointer grab.

Libraries

Enhanced X-Windows Library (**libX11.a**)

FORTRAN 77 Library (**libxfx.a**)

C Syntax

XChangeActivePointerGrab(*DisplayPtr*, *EventMask*, *CursorID*, *TimeStamp*)
Display **DisplayPtr*;
unsigned int *EventMask*;
Cursor *CursorID*;
Time *TimeStamp*;

FORTRAN Syntax

external fxchangeactivepointergrab
integer*4 *DisplayPtr*
integer*4 *EventMask*, *CursorID*, *TimeStamp*
call fxchangeactivepointergrab(*DisplayPtr*, *EventMask*, *CursorID*, *TimeStamp*)

Description

The **XChangeActivePointerGrab** subroutine changes the specified dynamic parameters if the pointer is actively grabbed by the client with a specified time no earlier than the last-pointer-grab time and no later than the current X Server time. The **XChangeActivePointerGrab** subroutine has no effect on the passive parameters of the **XGrabButton** subroutine.

Parameters

CursorID Specifies the cursor to be displayed or the value of **None**.

DisplayPtr Specifies the connection to the X Server.

EventMask Specifies the pointer events to be reported to the client. The *EventMask* parameter can be one of the following values:

ButtonPressMask	**ButtonReleaseMask**
EnterWindowMask	**LeaveWindowMask**
Button1MotionMask	**Button2MotionMask**
Button3MotionMask	**Button4MotionMask**
Button5MotionMask	**PointerMotionHintMask**
PointerMotionMask	**ButtonMotionMask**
KeymapStateMask	

TimeStamp Specifies the time in a timestamp, which is expressed in milliseconds, or the **CurrentTime** value.

Error Code

BadCursor

Implementation Specifics

This Xlib subroutine is part of AIXwindows Run Time Environment in AIXwindows Environment/6000.

Related Information

The **XGrabButton** subroutine.

The **ChangeActivePointerGrab** protocol request.

XChangeGC Subroutine

Purpose

Changes the components in the specified graphics context.

Libraries

Enhanced X-Windows Library (**libX11.a**)

FORTRAN 77 Library (**libXfx.a**)

C Syntax

XChangeGC(*DisplayPtr*, *GraphicsContext*, *ValueMaskChange*,*Values)*
Display **DisplayPtr*;
GC *GraphicsContext*;
unsigned long *ValueMaskChange;*
XGCValues **Values*;

FORTRAN Syntax

external fxchangegc
integer*4 *DisplayPtr*
integer*4 *GrahpicsContext*
integer*4 *ValueMaskChange*
integer*4 *Values*
call fxchangegc(*DisplayPtr*, *GraphicsContext*, *ValueMaskChange*, *Values*)

Description

The **XChangeGC** subroutine changes the components specified by the *ValueMaskChange* parameter 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.

Changing the clip-mask overrides any previous **XSetClipRectangles** request on the context. Changing the dash-offset or dash-list overrides any previous **XSetDashes** request on the context.

Components are verified according to mask values defined in the **< X11/X.h>** file.

Parameters

DisplayPtr — Specifies the connection to the X Server.

GraphicsContext — Specifies the graphics context.

ValueMaskChange — Specifies the components in the graphics context to be changed using information in the **XGCValues** structure. This value is the bitwise inclusive OR of one or more of the following valid **GC** component masks:

GCLineWidth	**GCLineStyle**
GCCapStyle	**GCJoinStyle**
GCFillStyle	**GCFillRule**

GCTile	**GCStipple**
GCTileStipXOrigin	**GCTileStipYOrigin**
GCFont	**GCSubwindowMode**
GCClipXOrigin	**GCGraphicsExposures**
GCClipYOrigin	**GCClipMask**
GCDashOffset	**GCDashList**
GCArcMode	**GCFunction**
GCForeground	**GCBackground**
GCPlanemask	

Values Specifies a pointer to the **XGCValues** structure.

Error Codes

BadAlloc

BadFont

BadGC

BadImplementation

BadMatch

BadPixmap

BadValue

Implementation Specifics

This Xlib subroutine is part of AIXwindows Run Time Environment in AIXwindows Environment/6000.

Related Information

The **XGCValues** data structure.

The **ChangeGC** protocol request.

XChangeKeyboardControl Subroutine

Purpose

Changes keyboard settings.

Libraries

Enhanced X-Windows Library (**libX11.a**)

FORTRAN 77 Library (**libXfx.a**)

C Syntax

XChangeKeyboardControl(*DisplayPtr*, *ValueMask*, *Values*)
Display **DisplayPtr*;
unsigned long *ValueMask*;
XKeyboardControl **Values*;

FORTRAN Syntax

external fxchangekeyboardcontrol
integer*4 *DisplayPtr*
integer*4 *ValueMask*, *Values*
call fxchangekeyboardcontrol(*DisplayPtr*, *ValueMask*, *Values*)

Description

The **XChangeKeyboardControl** subroutine controls the keyboard characteristics defined by the **XKeyboardControl** structure.

Parameters

DisplayPtr Specifies the connection to the X Server.

ValueMask 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. Each 1 bit in this mask specifies that the corresponding field in the **XKeyboardControl** structure is to be changed. The following values can be OR'd together in the *ValueMask* parameter:

KBKeyClickPercent	**KBBellPercent**
KBBellPitch	**KBBellDuration**
KBLed	**KBLedMode**
KBKey	**KBAutoRepeatMode**

Values Specifies a pointer to the **XKeyboardControl** structure.

Error Codes

BadImplementation

BadMatch

BadValue

Implementation Specifics

This Xlib subroutine is part of AIXwindows Run Time Environment in AIXwindows Environment/6000.

Related Information

The **XKeyboardControl** data structure.

The **ChangeKeyboardControl** protocol request.

XChangeKeyboardMapping Subroutine

Purpose

Changes the mapping of key symbols to key codes.

Libraries

Enhanced X-Windows Library (**libX11.a**)

FORTRAN 77 Library (**libXfx.a**)

C Syntax

XChangeKeyboardMapping(*DisplayPtr*,*FirstKeycode*,
KeysymsPerKeycode, *Keysyms*,
NumberCodes)

Display **DisplayPtr*;
int *FirstKeycode*;
int *KeysymsPerKeycode*;
KeySym **Keysyms*;
int *NumberCodes*;

FORTRAN Syntax

external fxchangekeyboardcontrol
integer*4 *DisplayPtr*
integer*4 *FirstKeycode*
integer*4 *KeysymsPerKeycode*
integer*4 *Keysyms*, *NumberCodes*
call fxchangekeyboardmapping(*DisplayPtr*, *FirstKeycode*,
KeysymsPerKeycode, *Keysyms*,
NumberCodes)

Description

The **XChangeKeyboardMapping** subroutine defines the key symbols for the specified number of key codes starting with the first key code indicated in the *FirstKeycode* parameter. The symbols for key codes outside this range remain unchanged.

The number of elements in the *Keysyms* parameter must be a multiple of the *KeysymsPerKeycode* parameter. Otherwise, an error is generated.

The specified *FirstKeycode* parameter must be greater than or equal to *MinKeycode* parameter returned by **XDisplayKeycodes**.

In addition, the following expression must be less than or equal to *MaxKeycode* parameter as returned by **XDisplayKeycodes**:

```
FirstKeycode + NumberCodes - 1
```

The value of **KeySym** number N, counting from 0, for the K key code has the following index in the *Keysyms* parameter, counting from 0:

```
(K - FirstKeycode) * KeysymsPerKeycode + N
```

The specified *KeysymsPerKeycode* parameter can be chosen arbitrarily by the client to be large enough to hold all desired symbols. The **KeySym** value of the **NoSymbol** should be used for undefined elements in individual key codes. The **NoSymbol** value can be displayed in nontrailing positions of the effective list for a key code.

The number of elements in the *Keysyms* parameter list must be a multiple of those in the *KeysymsPerKeycode* parameter. Otherwise, an error is generated.

The **XChangeKeyboardMapping** subroutine generates a **MappingNotify** event. The X Server does not need to interpret this mapping, but should merely store it for reading and writing by clients.

Parameters

DisplayPtr	Specifies the connection to the X Server.
FirstKeycode	Specifies the first key code to be changed.
KeysymsPerKeycode	Specifies the key symbols to be used.
Keysyms	Specifies a pointer to an array of key symbols.
NumberCodes	Specifies the number of key codes that are to be changed.

Error Codes

BadAlloc

BadImplementation

BadLength

BadValue

Implementation Specifics

This Xlib subroutine is part of AIXwindows Run Time Environment in AIXwindows Environment/6000.

Related Information

The **ChangeKeyboardMapping** protocol request.

XChangePointerControl Subroutine

Purpose

Changes the rate of acceleration in the movement of a pointer device.

Libraries

Enhanced X-Windows Library (**libX11.a**)

FORTRAN 77 Library (**libXfx.a**)

C Syntax

XChangePointerControl(*DisplayPtr*, *DoAccelerate*, *DoThreshold*,
AccelerationNumerator,
AccelerationDenominator, *Threshold***)**
Display **DisplayPtr*;
Bool *DoAccelerate*, *DoThreshold*;
int *AccelerationNumerator*, *AccelerationDenominator*;
int *Threshold*;

FORTRAN Syntax

external fxchangepointercontrol
integer*4 *DisplayPtr*
integer*4 *Doaccelerator*, *Dothreshold*
integer*4 *Accelerationnumerator*, *Accelerationdenominator*, *Threshold*
call fxchangepointereturncontrol(*DisplayPtr*, *Doaccelerator*, *Dothreshold*,
Accelerationnumerator, *Accelerationdenominator*,
*Threshold***)**

Description

The **XChangePointerControl** subroutine 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 can be rounded arbitrarily by the X Server. Acceleration takes effect after the pointer moves more than the threshold number of pixels, and only takes effect beyond the value in the *Threshold* parameter. To restore the default, set the value to **–1**.

The values of the *DoAccelerate* and *DoThreshold* parameters must be nonzero in order to set the pointer values. Otherwise, the parameters are not changed.

Parameters

AccelerationDenominator	Specifies the denominator for the acceleration multiplier.
AccelerationNumerator	Specifies the numerator for the acceleration multiplier.
DisplayPtr	Specifies the connection to the X Server.
DoAccelerate	Specifies a Boolean value that controls whether the values for the *AccelerationNumerator* or *AccelerationDenominator* parameter are used.
DoThreshold	Specifies a Boolean value that controls whether the value for the *Threshold* parameter is used.

Threshold	Specifies the acceleration threshold.

Error Codes

BadImplementation

BadValue

Implementation Specifics

This Xlib subroutine is part of AIXwindows Run Time Environment in AIXwindows Environment/6000.

Related Information

The **ChangePointerControl** protocol request.

XChangeProperty Subroutine

Purpose

Changes the property for a specified window.

Libraries

Enhanced X-Windows Library (**libX11.a**)

FORTRAN 77 Library (**libXfx.a**)

C Syntax

XChangeProperty(*DisplayPtr*, *WindowID*, *Property*, *Type*, *Format*, *Mode*, *Data*, *NumberElements*)
Display **DisplayPtr*;
Window *WindowID*;
Atom *Property*, *Type*;
int *Format*;
int *Mode*;
unsigned char **Data*;
int *NumberElements*;

FORTRAN Syntax

external fxchangeproperty
integer*4 *DisplayPtr*
integer*4 *WindowID*
integer*4 *Property*, *Type*, *Format*, *Mode*
integer*4 *Data*
integer*4 *NumberElements*
call fxchangeproperty(*DisplayPtr*, *WindowID*, *Property*, *Type*, *Format*, *Mode*, *Data*, *NumberElements*)

Description

The **XChangeProperty** subroutine changes the property for a specified window. If the property does not exist, it adds the property. The **XChangeProperty** subroutine causes the X Server to generate a **PropertyNotify** event on a specified window. The lifetime of a property is not tied to the client. Properties are not deleted like resources; they remain until explicitly deleted, or the window is eliminated, or the server is reset.

The X Server does not interpret the *Type* parameter, but simply passes it back to an application that later calls the **XGetWindowProperty** subroutine.

The *Format* parameter allows the X Server to correctly perform value-swap operations for the 8-bit, 16-bit, and 32-bit values. If the *Format* parameter value is 16-bit or 32-bit, it must explicitly cast the data pointer to a **(char *)** data type when it calls the **XChangeProperty** subroutine.

Parameters

Data	Specifies the property data.
DisplayPtr	Specifies the connection to the X Server.
Format	Specifies the data format as a list of 8-bit, 16-bit, or 32-bit quantities.

Mode — Specifies the mode. The *Mode* parameter can be set to:

- **PropModeReplace** — The **XChangeProperty** subroutine discards the previous *Property* parameter value and stores the new data.
- **PropModePrepend** — The *Type* and *Format* parameters must match the existing *Property* parameter. If the *Property* parameter is undefined, it is treated as matching the correct *Type* and *Format* parameters, with zero-length data. The **XChangeProperty** subroutine inserts the data before the beginning of the existing data.
- **PropModeAppend** — The *Type* and *Format* parameters must match the existing *Property* parameter. If the *Property* parameter is undefined, it is treated as matching the correct *Type* and *Format* parameters, with zero-length data. The **XChangeProperty** subroutine inserts the data after the end of the existing data.

NumberElements — Specifies the number of elements of the specified data format.

Property — Specifies the property atom.

Type — Specifies the property type.

WindowID — Specifies the window ID for the window whose property is to be changed.

Error Codes

BadAlloc

BadAtom

BadImplementation

BadMatch

BadValue

BadWindow

Implementation Specifics

This Xlib subroutine is part of AIXwindows Run Time Environment in AIXwindows Environment/6000.

Related Information

The **ChangeProperty** protocol request.

The **XGetWindowProperty** subroutine.

XChangeSaveSet Subroutine

Purpose

Adds or removes a window from the save-set of the client.

Libraries

Enhanced X-Windows Library (**libX11.a**)

FORTRAN 77 Library (**libXfx.a**)

C Syntax

XChangeSaveSet(*DisplayPtr*, *WindowID*, *ChangeMode***)**
**Display ****DisplayPtr*;
Window *WindowID*;
int *ChangeMode*;

FORTRAN Syntax

external fxchangesaveset
integer*4 *Display*
integer*4 *WindowID*, *ChangeMode*
call fxchangesaveset(*DisplayPtr*, *WindowID*, *ChangeMode***)**

Description

The **XChangeSaveSet** subroutine adds or removes a subwindow from the save-set of the client depending on the *ChangeMode* parameter. The window specified must be one created by another client.

Parameters

ChangeMode — Specifies the mode. The *ChangeMode* parameter can be set to:

- **SetModeInsert** — The **XChangeSaveSet** subroutine adds the window to the client save-set.
- **SetModeDelete** — The **XChangeSaveSet** subroutine deletes the window from the client save-set. The X Server automatically removes windows from the save-set when they are destroyed.

DisplayPtr — Specifies the connection to the X Server.

WindowID — Specifies the window ID of the window where child windows are to be added to the client's save-set.

Error Codes

BadImplementation

BadMatch

BadValue

BadWindow

Implementation Specifics

This Xlib subroutine is part of AIXwindows Run Time Environment in AIXwindows Environment/6000.

Related Information

The **ChangeSaveSet** protocol request.

The **XAddToSaveSet** subroutine, **XRemoveFromSaveSet** subroutine.

XChangeWindowAttributes Subroutine

Purpose

Changes one or more window attributes.

Libraries

Enhanced X-Windows Library (**libX11.a**)

FORTRAN 77 Library (**libXfx.a**)

C Syntax

XChangeWindowAttributes(*DisplayPtr*, *WindowID*, *ValueMask*, *Attributes*)
Display **DisplayPtr*;
Window *WindowID*;
unsigned long *ValueMask*;
XSetWindowAttributes **Attributes*;

FORTRAN Syntax

external fxchangewindowattributes
integer*4 *DisplayPtr*
integer*4 *WindowID*
integer*4 *ValueMask*
integer*4 *Attributes*
call fxchangewindowattributes(*DisplayPtr*, *WindowID*, *ValueMask*, *Attributes*)

Description

The **XChangeWindowAttributes** subroutine uses the window attributes in the **XSetWindowAttributes** data structure to change the specified window parameters depending on the *Valuemask* parameter. When using the **XChangeWindowAttributes** subroutine, note the following:

- Changing the background does not cause the window contents to be changed. Use the **XClearWindow** subroutine to repaint the window and the background.
- Setting the border, or changing the background such that the border tile origin changes, causes the border to be repainted.
- Changing the background of a root window to the **None** or **ParentRelative** value restores the default background pixmap.
- Changing the border of a root window to the **CopyFromParent** value restores the default border pixmap.
- Changing the *win_gravity* attribute does not affect the current position of the window.
- Changing the *backing_store* attribute of an obscured window to the **WhenMapped** or **Always** value may have no immediate effect.
- Changing the *backing_planes*, *backing_pixel*, or *save_under* attributes of a mapped window may have no immediate effect.
- The event masks are maintained separately when multiple clients select input on the same window. When an event is generated, it will be reported to all interested clients.

However, only one client can select the **SubstructureRedirectMask**, **ResizeRedirectMask**, and **ButtonPressMask** event masks 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 while 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 because the colormap may not be installed.
- Windows should share colormaps whenever possible.
- Changing the cursor of a root window to the **None** value restores the default cursor.

Parameters

Attributes Specifies the parameters of the window that will be set at the time the specified window is created.

DisplayPtr Specifies the connection to the X Server.

ValueMask Specifies the defined window parameters. This mask is the bitwise inclusive OR of the valid parameter mask bits. If the *ValueMask* parameter is a value of **0**, other parameters are ignored, and are not referenced.

WindowID Specifies the window ID.

Error Codes

BadAccess

BadColor

BadCursor

BadImplementation

BadMatch

BadPixmap

BadValue

BadWindow

Implementation Specifics

This Xlib subroutine is part of AIXwindows Run Time Environment in AIXwindows Environment/6000.

Related Information

The **XSet WindowAttributes** data structure.

The **XChangeWindowAttributes** protocol request.

The **XClearWindow** subroutine, **XInstallColormap** subroutine, **XSetWindowBackground** subroutine, **XSetWindowBackgroundPixmap** subroutine, **XSetWindowBorder** subroutine, **XSetWindowBorderPixmap** subroutine.

XCheckIfEvent Subroutine

Purpose

Checks the event queue for a specified event without blocking.

Libraries

Enhanced X-Windows Library (**libX11.a**)

FORTRAN 77 Library (**libXfx.a**)

C Syntax

Bool XCheckIfEvent(*DisplayPtr*, *EventReturn*, *Predicate*, *Argument*)
Display **DisplayPtr*;
XEvent **EventReturn*;
Bool (**Predicate*)();
char **Argument*;

FORTRAN Syntax

integer*4 fxcheckifevent
external fxcheckifevent
integer*4 *DisplayPtr*
integer*4 *EventReturn*
integer*4 *Predicate*
character*256 *Argument*
integer*4 *ReturnCode*
ReturnCode = **fxcheckifevent**(*DisplayPtr*, *EventReturn*, *Predicate*, *Argument*)

external *FunctionName*
integer*4 *DisplayPtr*
integer*4 *Event*
integer*4 *Arguments*
call *FunctionName* (*DisplayPtr*, *Event*, *Arguments*)

Description

The **XCheckIfEvent** subroutine copies the event into the client-supplied **XEvent** structure when the predicate procedure finds a match. The **XCheckIfEvent** subroutine uses the following predicate procedure:

Bool (**predicate*) (*DisplayPtr*, *Event*, *Argument*)
Display **DisplayPtr*;
XEvent **Event*;
char **Argument*;

Parameters

Argument	Specifies the user-supplied parameter to be passed to the predicate procedure.
DisplayPtr	Specifies the connection to the X Server.
EventReturn	Copies the matched events 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 parameter.

Return Values

False	Indicates that the predicate procedure does not find a match. The output buffer is flushed, but events stored in the queue earlier are not discarded.
True	Indicates that the matched event is found; this event is removed from the queue.

Error Code

BadImplementation

Related Information

The **XIfEvent** subroutine, **XPeekIfEvent** subroutine.

Implementation Specifics

This Xlib subroutine is part of AIXwindows Run Time Environment in AIXwindows Environment/6000.

XCheckMaskEvent Subroutine

Purpose

Removes the next event that matches a specified mask, without blocking.

Libraries

Enhanced X-Windows Library (**libX11.a**)

FORTRAN 77 Library (**libXfx.a**)

C Syntax

Bool XCheckMaskEvent(*DisplayPtr*, *EventMask*, *EventReturn*)
**Display ****DisplayPtr*;
unsigned long *EventMask*;
**XEvent ****EventReturn*;

FORTRAN Syntax

integer*4 fxcheckmaskevent
external fxcheckmaskevent
integer*4 *DisplayPtr*
integer*4 *EventMask*, *EventReturn*
integer*4 *ReturnCode*
ReturnCode = **fxcheckmaskevent(***DisplayPtr*, *EventMask*, *EventReturn*)

Description

The **XCheckMaskEvent** subroutine searches first the event queue, and 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 the **True** value. The other events stored in the queue are not discarded.

Parameters

DisplayPtr	Specifies the connection to the X Server.
EventMask	Specifies the event mask which is the bitwise inclusive OR of one or more of the valid event mask bits.
EventReturn	Copies the associated structure of the matched event into this client-supplied structure.

Return Values

False	Indicates that the event requested is not in the queue. The **XCheckMaskEvent** subroutine flushes the output buffer and returns.
True	Indicates that the **XCheckMaskEvent** subroutine finds a match. It removes that event, copies it into the specified **XEvent** structure, and returns this value.

Error Code

BadImplementation

Implementation Specifics

This Xlib subroutine is part of AIXwindows Run Time Environment in AIXwindows Environment/6000.

XCheckTypedEvent Subroutine

Purpose

Gets the next event that matches the event type.

Libraries

Enhanced X-Windows Library (**libX11.a**)

FORTRAN 77 Library (**libXfx.a**)

C Syntax

int XCheckTypedEvent(*DisplayPTr*, *EventType*, *EventReturn***)**
**Display ****DisplayPtr*;
int *EventType*;
**XEvent ****EventReturn*;

FORTRAN Syntax

integer*4 fxchecktypedevent
external fxchecktypedevent
integer*4 *DisplayPtr*
integer*4 *EventType*, *EventReturn*
integer*4 *ReturnCode*
ReturnCode = **fxchecktypedevent(***DisplayPtr*, *EventType*, *EventReturn***)**

Description

The **XCheckTypedEvent** subroutine searches the event queue, and 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 the **True** value. The other events in the queue are not discarded.

Parameters

DisplayPtr	Specifies the connection to the X Server.
EventReturn	Returns the matched events associated structure into this client-supplied structure.
EventType	Specifies the event type to be compared.

Return Values

False	If the event is not available.
True	If the **XCheckTypedEvent** subroutine finds a match.

Error Code

BadImplementation

Implementation Specifics

This Xlib subroutine is part of AIXwindows Run Time Environment in AIXwindows Environment/6000.

Related Information

The **XEvent** subroutine.

XCheckTypedWindowEvent Subroutine

Purpose

Gets the next event for a specified window.

Libraries

Enhanced X-Windows Library (**libX11.a**)

FORTRAN 77 Library (**libXfx.a**)

C Syntax

int XCheckTypedWindowEvent(*DisplayPtr*, *WindowID*, *EventType*, *EventReturn***)**
Display ******DisplayPtr*;
Window *WindowID*;
int *EventType*;
XEvent ******EventReturn*;

FORTRAN Syntax

integer*4 fxchecktypedwindowevent
external fxchecktypedwindowevent
integer*4 *DisplayPtr*
integer*4 *WindowID*, *EventType*, *EventReturn*
integer*4 *ReturnCode*
ReturnCode = **fxchecktypedwindowevent(***DisplayPtr*, *WindowID*, *EventType*, *EventReturn***)**

Description

The **XCheckTypedWindowEvent** subroutine searches the event queue, and then any events available on the server connection, for the first event that matches the specified type and window. When it finds a match, the other events in the queue are not discarded.

Parameters

DisplayPtr	Specifies the connection to the X Server.
EventReturn	Copies the associated structure of the matched events into this client-supplied structure.
EventType	Specifies the event type to be compared.
WindowID	Specifies the window ID.

Return Values

False	Indicates that the event is not available. The **XCheckTypedWindowEvent** subroutine flushes the output buffer and returns this value.
True	Indicates that the **XCheckTypedWindowEvent** subroutine finds a match. It removes the event from the queue, copies it into the specified **XEvent** structure, and returns this value.

Error Code

BadImplementation

Implementation Specifics

This Xlib subroutine is part of AIXwindows Run Time Environment in AIXwindows Environment/6000.

Related Information

The**XEvent** data structure.

XCheckWindowEvent Subroutine

Purpose

Removes the next event that matches the specified window and mask, without blocking.

Libraries

Enhanced X-Windows Library (**libX11.a**)

FORTRAN 77 Library (**libXfx.a**)

C Syntax

Bool XCheckWindowEvent(*DisplayPtr*, *WindowID*, *EventMask*, *EventReturn*)
Display **DisplayPtr*;
Window *WindowID*;
int *EventMask*;
XEvent **EventReturn*;

FORTRAN Syntax

integer*4 fxcheckwindowevent
external fxcheckwindowevent
integer*4 *DisplayPtr*, *WindowID*
integer*4 *EventMask*, *EventReturn*
integer*4 *ReturnCode*
ReturnCode = **fxcheckwindowevent(***DisplayPtr*, *WindowID*, *EventMask*, *EventReturn*)

Description

The **XCheckWindowEvent** subroutine searches the event queue, and then the events available on the server, for the first event that matches the specified window and event mask. When it finds a match, the other events stored in the queue are not discarded.

Parameters

DisplayPtr	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	Copies the associated structure of the matched event into this client-supplied structure.
WindowID	Specifies the window ID.

Return Values

False	Indicates that the event is not available. The **XCheckWindowEvent** subroutine flushes the output buffer.
True	Indicates that the XCheckWindowEvent subroutine finds a match. It removes the event and copies it into the specified **XEvent** structure.

Error Code

BadImplementation

Implementation Specifics

This Xlib subroutine is part of AIXwindows Run Time Environment in AIXwindows Environment/6000.

Related Information

The **XEvent** data structure.

XCirculateSubwindows Subroutine

Purpose

Circulates a subwindow up or down.

Libraries

Enhanced X-Windows Library (**libX11.a**)

FORTRAN 77 Library (**libXfx.a**)

C Syntax

XCirculateSubwindows(*DisplayPtr*, *WindowID*, *Direction*)
Display **DisplayPtr*;
Window *WindowID*;
int *Direction*;

FORTRAN Syntax

external fxcirculatesubwindows
integer*4 *DisplayPtr*
integer*4 *WindowID*
integer*4 *Direction*
call fxcirculatesubwindows(*DisplayPtr*, *WindowID*, *Direction*)

Description

The **XCirculateSubwindows** subroutine circulates the specified subwindow. If another client has selected **SubstructureRedirectMask**, a **CirculateRequest** event is generated, and no further processing is performed. Otherwise, the window is raised or lowered as specified.

If the window is restacked, the X Server generates a **CirculateNotify** event.

The *Direction* parameter can be the **RaiseLowest** or **LowerHighest** value.

- If the **RaiseLowest** value is specified, the **XCirculateSubwindows** subroutine raises the lowest mapped child window (if any) that is occluded by another child window to the top of the stack.
- If the **LowerHighest** value is specified, the **XCirculateSubwindows** subroutine lowers the highest mapped child window (if any) that occludes another child window to the bottom of the stack.

Exposure processing is performed on formerly obscured windows.

Parameters

Direction	Specifies the direction for circulating the window.
DisplayPtr	Specifies the connection to the X Server.
WindowID	Specifies the window ID of the window to be circulated.

Error Codes

BadImplementation

BadValue

BadWindow.

Implementation Specifics

This Xlib subroutine is part of AIXwindows Run Time Environment in AIXwindows Environment/6000.

Related Information

The **CirculateWindow** protocol request.

XCirculateSubwindowsDown Subroutine

Purpose

Lowers the highest mapped child of the specified window.

Libraries

Enhanced X-Windows Library (**libX11.a**)

FORTRAN 77 Library (**libXfx.a**)

C Syntax

XCirculateSubwindowsDown(*DisplayPtr*, *WindowID*)
Display **DisplayPtr*;
Window *WindowID*;

FORTRAN Syntax

external fxcirculatesubwindowsdown
integer*4 *DisplayPtr*
integer*4 *WindowID*
call fxcirculatesubwindowsdown(*DisplayPtr*, *WindowID*)

Description

The **XCirculateSubwindowsDown** subroutine lowers the highest mapped child of the specified window that partially or completely occludes another child window. Unobscured child windows are not affected.

Parameters

DisplayPtr Specifies the connection to the X Server.

WindowID Specifies the window ID for the window to be lowered.

Error Codes

BadImplementation

BadWindow

Implementation Specifics

This Xlib subroutine is part of AIXwindows Run Time Environment in AIXwindows Environment/6000.

Related Information

The **CirculateWindow** protocol request.

XCirculateSubwindowsUp Subroutine

Purpose

Raises the lowest mapped child of the specified window.

Libraries

Enahnced X-Windows Library (**libX11.a**)

FORTRAN 77 Library (**libXfx.a**)

C Syntax

XCirculateSubwindowsUp(*DisplayPtr*, *WindowID*)
**Display ****DisplayPtr*;
Window *WindowID*;

FORTRAN Syntax

external fxcirculatesubwindowsup
integer*4 *DisplayPtr*
integer*4 *WindowID*
call fxcirculatesubwindowsup(*DisplayPtr*, *WindowID*)

Description

The **XCirculateSubwindowsUp** subroutine raises the lowest mapped child of the specified window that is partially or completely occluded by another child window. Unobscured child windows are not affected.

Parameters

DisplayPtr	Specifies the connection to the X Server.
WindowID	Specifies the window ID for the window to be raised.

Error Codes

BadImplementation

BadWindow

Implementation Specifics

This Xlib subroutine is part of AIXwindows Run Time Environment in AIXwindows Environment/6000.

Related Information

The **CirculateWindow** protocol request.

XClearArea Subroutine

Purpose

Clears a rectangular area of the specified window.

Libraries

Enhanced X-Windows Library (**libX11.a**)

FORTRAN 77 Library (**libXfx.a**)

C Syntax

XClearArea(*DisplayPtr*, *WindowID*, *X*, *Y*, *Width*, *Height*, *Exposures*)
Display **DisplayPtr*;
Window *WindowID*;
int *X*, *Y*;
unsigned int *Width*, *Height*;
Bool *Exposures*;

FORTRAN Syntax

external fxcleararea
integer*4 *DisplayPtr*
integer*4 *WindowID*
integer*4 *X*, *Y*
integer*4 *Width, Height*
integer*4 *Exposures*
call fxcleararea(*DisplayPtr*, *WindowID*, *X*, *Y*, *Width*, *Height*, *Exposures*)

Description

The **XClearArea** subroutine paints a rectangular area in the specified window according to specified dimensions with the background pixel or pixmap of the window.

- If the *Width* parameter is a value of **0**, it is replaced with the current width of the window minus the *X* parameter.
- If the *Height* parameter is a value of **0**, it is replaced with the current height of the window minus the *Y* parameter.
- If the window has a defined background tile, the rectangle is filled with this tile.
- If the window has the background value of **None**, the contents of the window are not changed.

In both cases, if the *Exposures* parameter is the value of **True**, one or more **Expose** events are generated for regions of the rectangle that are either visible or in a backing store.

The **XClearArea** subroutine cannot be used with an **InputOnly** class window.

Parameters

DisplayPtr	Specifies the connection to the X server.
Exposures	Specifies a Boolean value of **True** or **False**.
Height	Specifies the height dimension of the rectangle.
Width	Specifies the width dimension of the rectangle.
WindowID	Specifies the window ID.
X	Specifies the x coordinate at the upper-left corner of the rectangle.
Y	Specifies the x coordinate at the upper-left corner of the rectangle.

Error Codes

BadImplementation

BadMatch

BadValue

BadWindow

Implementation Specifics

This Xlib subroutine is part of AIXwindows Run Time Environment in AIXwindows Environment/6000.

Related Information

The **ClearArea** protocol request

XClearWindow Subroutine

Purpose

Clears the entire window.

Libraries

Enhanced X-Windows Library (**libX11.a**)

FORTRAN 77 Library (**libXfx.a**)

C Syntax

XClearWindow(*DisplayPtr*, *WindowID***)**
**Display ****DisplayPtr*;
Window *WindowID*;

FORTRAN Syntax

external fxclearwindow
integer*4 *DisplayPtr*
integer*4 *WindowID*
call fxclearwindow(*DisplayPtr*, *WindowID***)**

Description

The **XClearWindow** subroutine clears the entire area in the window specified.

- If the window has a defined background tile, the rectangle is tiled with a plane mask of all 1s and the **GXCopy** display function.
- If the window background has a value of **None**, the contents of the window are not changed.

The **XClearWindow** subroutine cannot be used with an **InputOnly** class window.

Parameters

DisplayPtr	Specifies the connection to the X Server.
WindowID	Specifies the window ID of the window to be cleared.

Error Codes

BadImplementation

BadMatch

BadValue

BadWindow

Implementation Specifics

This Xlib subroutine is part of AIXwindows Run Time Environment in AIXwindows Environment/6000.

Related Information

The **ClearArea** protocol request.

XClipBox Subroutine

Purpose

Generates the smallest rectangle enclosing a specified region.

Libraries

Enhanced X-Windows Library (**libX11.a**)

FORTRAN 77 Library (**libXfx.a**)

C Syntax

XClipBox(*RegionPtr*, *RectangleReturn***)**
Region *RegionPtr*;
XRectangle **RectangleReturn*;

FORTRAN Syntax

external fxclipbox
integer*4 *RegionPtr*, *RectangleReturn*
call fxclipbox(*RegionPtr*, *RectangleReturn***)**

Description

The **XClipBox** subroutine generates the smallest enclosing rectangle in the *RectangleReturn* parameter. The opaque type **Region** value is defined in the **<X11/Xutil.h>** file.

Parameters

RectangleReturn	Specifies the rectangle in which the smallest enclosing rectangle is generated.
RegionPtr	Specifies the region where the rectangle is located.

Error Code

BadImplementation

Implementation Specifics

This Xlib subroutine is part of AIXwindows Run Time Environment in AIXwindows Environment/6000.

Related Information

The **XRectangle** data structure.

XCloseDisplay Subroutine

Purpose

Closes a display.

Libraries

Enhanced X-Windows Library (**libX11.a**)

FORTRAN 77 Library (**libXfx.a**)

C Syntax

XCloseDisplay(*DisplayPtr***)**
Display ***DisplayPtr;*

FORTRAN Syntax

external fxclosedisplay
integer*4 *DisplayPtr*
call fxclosedisplay(*DisplayPtr***)**

Description

The **XCloseDisplay** subroutine closes or disconnects a display from the X Server. It destroys all windows, resource IDs (**Window**, **Font**, **Pixmap**, **Colormap**, **Cursor**, and **GContext**), or other graphic contexts that the client created on the display device, unless the close-down mode of the resource is changed. These windows, resource IDs, and other graphic contexts should not be referenced again.

The **XCloseDisplay** subroutine discards any output requests that are buffered but not sent.

Parameter

DisplayPtr Specifies the connection to the X Server.

Error Codes

BadGC

BadImplementation

Implementation Specifics

This Xlib subroutine is part of AIXwindows Run Time Environment in AIXwindows Environment/6000.

XConfigureWindow Subroutine

Purpose

Configures a window for size, position, border, and stacking order.

Libraries

Enhanced X-Windows Library (**libX11.a**)

FORTRAN 77 Library (**libXfx.a**)

C Syntax

XConfigureWindow(*DisplayPtr*, *WindowID*, *ValueMask*, *Values*)
Display **DisplayPtr*;
Window *WindowID*;
unsigned int *ValueMask*;
XWindowChanges **Values*;

FORTRAN Syntax

external fxconfigurewindow
integer*4 *DisplayPtr*
integer*4 *WindowID*
integer*4 *ValueMask*
integer*4 *Values*
call fxconfigureWindow(*DisplayPtr*, *WindowID*, *ValueMask*, *Values*)

Description

The **XConfigureWindow** subroutine configures the size, position, border and stacking order of a window using the values specified in the **XWindowChanges** data structure. It takes any unspecified values from the existing geometry of the window. The stacking order of the window is controlled by the parameters.

Parameters

DisplayPtr	Specifies the connection to the X Server.
ValueMask	Specifies the values to be set in the *Values* parameter. This mask is the bitwise inclusive OR of the valid configure window values bits.
Values	Specifies a pointer to the **XWindowChanges** data structure.
WindowID	Specifies the window ID of the window to be reconfigured.

Error Codes

BadImplementation

BadMatch

BadValue

BadWindow

Implementation Specifics

This Xlib subroutine is part of AIXwindows Run Time Environment in AIXwindows Environment/6000.

Related Information

The **XWindowChanges** data structure.

The **ConfigureWindow** protocol request.

XConvertSelection Subroutine

Purpose

Converts a selection.

Libraries

Enhanced X-Windows Library (**libX11.a**)

FORTRAN 77 Library (**libXfx.a**)

C Syntax

XConvertSelection(*DisplayPtr*, *Selection*, *Target*, *Property*, *Requestor*, *TimeStamp*)
Display **DisplayPtr*;
Atom *Selection*, *Target*;
Atom *Property*;
Window *Requestor*;
Time *TimeStamp*;

FORTRAN Syntax

external fxconvertselection
integer*4 *DisplayPtr*
integer*4 *Selection*, *Target*
integer*4 *Property*, *Requestor*, *TimeStamp*
call fxconvertselection(*DisplayPtr*, *Selection*, *Target*, *Property*, *Requestor*, *TimeStamp*)

Description

The **XConvertSelection** subroutine requests that the specified selection be converted to the specified target type.

- If the specified selection has an owner, the X Server sends a **SelectionRequest** event to that owner.
- If the specified selection does not have an owner, the X Server generates a **SelectionNotify** event, with the property set to the **None** value, to the requestor .

In both events, the parameters are passed unchanged.

Parameters

DisplayPtr	Specifies the connection to the X Server.
Property	Specifies the property atom.
Requestor	Specifies the window ID of the window initiating the request.
Selection	Specifies the selection atom.
Target	Specifies the target atom.
TimeStamp	Specifies the time in either a timestamp, expressed in milliseconds, or in the **CurrentTime** value.

Error Codes

BadAtom

BadImplementation

BadWindow

Implementation Specifics

This Xlib subroutine is part of AIXwindows Run Time Environment in AIXwindows Environment/6000.

Related Information

The **SelectionNotify** event, **SelectionRequestEvent**.

The **ConvertSelection** protocol request.

XCopyArea Subroutine

Purpose

Copies the drawable area between drawables of the same root and depth.

Libraries

Enhanced X-Windows Library (**libX11.a**)

FORTRAN 77 Library (**libXfx.a**)

C Syntax

XCopyArea(*DisplayPtr*, *Source*, *Destination*, *GraphicsContext*, *SourceX*, *SourceY*, *Width*, *Height DestinationX*, *DestinationY*)
Display **DisplayPtr*;
Drawable *Source*, *Destination*;
GC *GraphicsContext*;
int *SourceX*, *SourceY*;
unsigned int *Width*, *Height*;
int *DestinationX*, *DestinationY*;

FORTRAN Syntax

external fxcopyarea
integer*4 *DisplayPtr*
integer*4 *Source*, *Source*, *GraphicsContext*
integer*4 *SourceX*, *SourceY*, *Width*, *Height*
integer*4 *DestinationX*, *DestinationY*
call fxcopyarea(*DisplayPtr*, *Source*, *Destination*, *GraphicsContext*, *SourceX*, *SourceY*, *Width*, *Height*, *DestinationX*, *DestinationY*)

Description

The **XCopyArea** subroutine copies an area of the specified drawable to another drawable. It combines the source rectangle specified by the *Source* parameter with the destination rectangle specified by the *Destination* parameter. The rectangles specified by these two parameters must have the same root and depth.

The regions of the source rectangle that are obscured and have not been retained in a *BackingStore* parameter are not copied. Specificed regions outside the boundaries of the source drawable are also not copied. Instead, the following occurs on all corresponding destination regions that are either visible or retained in a *BackingStore* parameter.

- If the destination rectangle is a window with a background other than the value of **None**, the corresponding regions of the destination are tiled with a *plane_mask* field of all 1s and with the **GXcopy** display function of that background.
- If the *graphics_exposures* field in the **GC** is the value of **True**, then the **GraphicsExpose** events for all corresponding destination regions are required.
- If the *graphics_exposures* field in the **GC** is the value of **True**, but no regions are exposed, a **NoExpose** event is generated. By default, the *graphics_exposures* field is the vlaue of **True** in new **GC** events.

The **XCopyArea** subroutine uses the *function, plane_mask, subwindow_mode, graphics_exposures, clip_x_origin, clip_y_origin,* and *clip_mask* graphics context fields.

Parameters

Destination	Specifies the destination rectangle to be combined with the source rectangle.
DestinationX	Specifies the x coordinate of the destination rectangle relative to its origin, at the upper-left corner of the destination rectangle.
DestinationY	Specifies the y coordinate of the destination rectangle relative to its origin, at the upper-left corner of the destination rectangle.
DisplayPtr	Specifies the connection to the X Server.
GraphicsContext	Specifies the graphics context.
Height	Specifies the height of both the source and destination rectangles.
Source	Specifies the source rectangle to be combined with the destination rectangle.
SourceX	Specifies the x coordinate of the source rectangle relative to its origin (the upper-left corner).
SourceY	Specifies the y coordinate of the source rectangle relative to its origin (the upper-left corner).
Width	Specifies the width of both the source and destination rectangles.

Error Codes

BadDrawable

BadGC

BadImplementation

BadMatch

Implementation Specifics

This Xlib subroutine is part of AIXwindows Run Time Environment in AIXwindows Environment/6000.

Related Information

The **CopyArea** protocol request.

The **XSetGraphicsExposures** subroutine.

XCopyColormapAndFree Subroutine

Purpose

Creates a new colormap from a previously shared colormap.

Libraries

Enhanced X-Windows Library (**libX11.a**)

FORTRAN 77 Library (**libXfx.a**)

C Syntax

Colormap XCopyColormapAndFree(*DisplayPtr*, *ColormapID*)
**Display ****DisplayPtr*
Colormap *ColormapID*

FORTRAN Syntax

integer*4 fxcopycolormapandfree
external fxcopycolormapandfree
integer*4 *DisplayPtr*
integer*4 *ColormapID*
integer*4 *ColormapIDReturned*
ColormapIDReturned **= fxcopycolormapandfree(***DisplayPtr*, *ColormapID*)

Description

The **XCopyColormapAndFree** subroutine creates a new colormap when the allocation of colormap entries from a previously-shared colormap was unsuccessful due to resource exhaustion. The **XCopyColormapAndFree** subroutine does the following:

- Creates a colormap of the same visual type for the same screen as the specified colormap and returns the new colormap ID.
- Moves all of the existing client allocation from the specified colormap to the new colormap. It keeps the color values of this allocation intact and frees these color entries in the specified colormap. It leaves the color values for other entries in the new colormap undefined.
- If the specified colormap was created by the client with the **AllocAll** parameter, the new colormap is also created with this parameter. The color values for all entries are then copied from the specified colormap, and all entries in the specified colormap are freed.
- If the specified colormap was not created by the client with the **AllocAll** parameter, the pixels and planes that are moved are those allocated by the client using the **XAllocColor**, **XAllocColorPlanes**, **XAllocColorCells**, or **XAllocNamedColor** subroutine.

Parameters

ColormapID	Specifies the colormap ID.
DisplayPtr	Specifies the connection to the X Server.

Error Codes

BadAlloc

BadColor

BadImplementation

Implementation Specifics

This Xlib subroutine is part of AIXwindows Run Time Environment in AIXwindows Environment/6000.

Related Information

The **CopyColormapAndFree** protocol request.

The **XCreateColormap** subroutine, **XFreeColormap** subroutine, **XFreeColors** subroutine.

XCopyGC Subroutine

Purpose

Copies components from a source graphics context to a destination graphics context.

Libraries

Enhanced X-Windows Library (**libX11.a**)

FORTRAN 77 Library (**libXfx.a**)

C Syntax

XCopyGC(*DisplayPtr*, *Source*, *ValueMaskCopy*,
Destination)
Display **DisplayPtr*;
GC *Source*;
unsigned long *ValueMaskCopy*;
GC *Destination*;

FORTRAN Syntax

external fxcopygc
integer*4 *DisplayPtr*
integer*4 *Source*
integer*4 *ValueMaskCopy*
integer*4 *Destination*
call fxcopygc(*DisplayPtr*, *Source*, *ValueMaskCopy*, *Destination*)

Description

The **XCopyGC** subroutine copies specified components from a source graphics context to a destination graphics context. Both graphics contexts must have the same root and depth.

Parameters

Destination	Specifies the destination graphics context.
DisplayPtr	Specifies the connection to the X Server.
Source	Specifies the components of the source graphics context.
ValueMaskCopy	Specifies the components in the source graphics context to be copied to the destination graphics context. This parameter is the bitwise inclusive OR of one or more of the valid **GC** component masks.

Error Codes

BadAlloc

BadGC

BadImplementation

BadMatch

BadValue

Implementation Specifics

This Xlib subroutine is part of AIXwindows Run Time Environment in AIXwindows Environment/6000.

Related Information

The **XCopyGC** protocol request.

XCopyPlane Subroutine

Purpose

Copies a single bit plane of a drawable.

Libraries

Enhanced X-Windows Library (**libX11.a**)

FORTRAN 77 Library (**libXfx.a**)

C Syntax

XCopyPlane(*DisplayPtr*, *Source*, *Destination*, *GraphicsContext*,*SourceX*,*SourceY*, *Width*, *Height*,*DestinationX*, *DestinationY*, *Plane*)
Display **DisplayPtr*;
Drawable *Source*, *Destination*;
GC *GraphicsContext*;
int *SourceX*, *SourceY*;
unsigned int *Width*, *Height*;
int *DestinationX*, *DestinationY*;
unsigned long *Plane*;

FORTRAN Syntax

external fxcopyplane
integer*4 *DisplayPtr*
integer*4 *Source*, *Destination*
integer*4 *GraphicsContext*
integer*4 *SourceX*, *SourceY*, *Width*, *Height*
integer*4 *DestinationX*, *DestinationY*, *Plane*
call fxcopyplane(*DisplayPtr*, *Source*, *Destination*, *GraphicsContext*, *SourceX*, *SourceY*, *Width*, *Height*, *DestinationX*, *DestinationY*, *Plane*)

Description

The **XCopyPlane** subroutine combines a single bit plane of the source rectangle with the specified destination rectangle. The rectangles must have the same root but not necessarily the same depth.

The **XCopyPlane** subroutine 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 when the bit plane in the source rectangle contains a 1 bit and it uses the background pixels when the bit plane in the source rectangle contains a 0 bit.

The equivalent of a **CopyArea** protocol request is performed, using 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* field value of **FillOpaqueStippled** for filling a destination rectangle.

If the *graphics_exposures* field in the **GC** is the value of **True**, the **GraphicsExpose** events for all corresponding destination regions are generated. If the *graphics_exposures* field is the value of **True**, but no regions are exposed, a **NoExpose** event is generated. By default, the *graphics_exposures* field has the value of **True** in new **GC**s.

The **XCopyPlane** subroutine uses the *function, plane_mask, foreground, background, subwindow_mode, graphics_exposures, clip_x_origin, clip_y_origin,* and *clip_mask* graphics context fields.

Parameters

Destination	Specifies the destination rectangle to be combined with the source rectangle.
DestinationX	Specifies the x coordinate of the upper-left corner of the destination rectangle relative to its origin.
DestinationY	Specifies the y coordinate of the upper-left corner of the destination rectangle relative to its origin.
DisplayPtr	Specifies the connection to the X server.
GraphicsContext	Specifies the graphics context.
Height	Specifies the height of both the source and destination rectangles.
Plane	Specifies the bit plane. Set one bit, for example: `0x01, 0x02, 0x04 (0x03 is illegal.)`
Source	Specifies the source rectangle to be combined with the destination rectangle.
SourceX	Specifies the x coordinate of the upper-left corner of the source rectangle relative to its origin.
SourceY	Specifies the y coordinate of the upper-left corner of the source rectangle relative to its origin.
Width	Specifies the width of both the source and destination rectangles.

Error Codes

BadDrawable

BadGC

BadImplementation

BadMatch

BadValue

Implementation Specifics

This Xlib subroutine is part of AIXwindows Run Time Environment in AIXwindows Environment/6000.

Related Information

The **CopyPlane** protocol request.

The **XSetGraphicsExposures** subroutine.

XCreateAssocTable Subroutine

Purpose

Returns a pointer to a newly created associate table.

Library

Enhanced X-Windows Library (**liboldX.a**)

Syntax

#include <X11/X10.h>
XAssocTable *XCreateAssocTable(*Size***)**
int *Size*;

Description

The **XCreateAssocTable** subroutine 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 pointer with the value of **NULL** is returned if there is an error allocating memory for the **XAssocTable** structure.

Note: This subroutine is in the **liboldX.a** library. Include this library in the compiler command to build your program. For example:

```
{compiler-option} -o samples samples.c -loldX -lX11
```

Parameter

Size — Specifies the number of buckets in the hash system of the **XAssocTable** structure.

Implementation Specifics

This Xlib subroutine is part of AIXwindows Run Time Environment in AIXwindows Environment/6000.

Related Information

The **XDestroyAssocTable** subroutine

XCreateBitmapFromData Subroutine

Purpose

Creates a bitmap from data.

Libraries

Enhanced X-Windows Library (**libX11.a**)

FORTRAN 77 Library (**libXfx.a**)

C Syntax

Pixmap XCreateBitmapFromData(*DisplayPtr*, *DrawableID*, *Data*, *Width*, *Height***)**
Display ****DisplayPtr*;**
Drawable *DrawableID***;**
char ****Data*;**
int *Width*, *Height***;**

FORTRAN Syntax

integer*4 fxcreatebitmapfromdata
external fxcreatebitmapfromdata
integer*4 *DisplayPtr*, *DrawableID*
integer*4 *Data*
integer*4 *Width*, *Height*
integer*4 *Pixmap*
Pixmap **= fxcreatebitmapfromdata(***DisplayPtr*, *DrawableID*, *Data*, *Width*, *Height***)**

Description

The **XCreateBitmapFromData** subroutine creates a bitmap from data stored in the program rather than reading a bitmap file that was written out by the **XWriteBitmapFile** subroutine. The **XCreateBitmapFromData** subroutine allows you to include a bitmap file written out by the **XWriteBitmapFile** subroutine without reading in the bitmap file.

For example, to include a gray bitmap, enter:

```
#include "gray.bitmap"
Pixmap bitmap;
bitmap = XCreateBitmapFromData(display, window, gray_bits,
                               gray_width, gray_height);
```

After this subroutine is completed, the **XFreePixmap** subroutine frees the bitmap.

Note: This subroutine is specific to C language programs using the **#include** file and following the AIX X-Windows version 2.1 format.

Parameters

DisplayPtr	Specifies the connection to the X Server.
DrawableID	Specifies the drawable.
Data	Specifies the location of the bitmap data.
Height	Specifies the height of the bitmap to be created.
Width	Specifies the width of the bitmap to be created.

Return Value

NULL	If insufficient working storage was allocated.
PixmapID	Resource ID of the created bitmap.

Error Codes

BadAlloc

BadImplementation

Implementation Specifics

This Xlib subroutine is part of AIXwindows Run Time Environment in AIXwindows Environment/6000.

Related Information

The **XCreatePixmapFromBitmapData** subroutine, **XWriteBitmapFile** subroutine, **XReadBitmapFile** subroutine.

XCreateColormap Subroutine

Purpose

Creates a colormap.

Libraries

Enhanced X-Windows Library (**libX11.a**)

FORTRAN 77 Library (**libXfx.a**)

C Syntax

Colormap XCreateColormap(*DisplayPtr*, *WindowID*, *VisualPtr*, *Allocate***)**
**Display ****DisplayPtr*;
Window *WindowID*;
**Visual ****VisualPtr*;
int *Allocate*;

FORTRAN Syntax

integer*4 fxcreatecolormap
external fxcreatecolormap
integer*4 *DisplayPtr*
integer*4 *WindowID*
integer*4 *VisualPtr*
integer*4 *Allocate*
integer*4 *Colormap*
Colormap = **fxcreatecolormap(***DisplayPtr*, *WindowID*, *VisualPtr*, *Allocate***)**

Description

The **XCreateColormap** subroutine creates a colormap of the specified visual type for the screen on which the window resides and associates the colormap ID with it. This subroutine operates on a **Visual** structure with members that contain information about the colormapping possible. The colormap for the specified window is not set; the specified window is used only to determine the screen.

The initial values of the colormap entries are undefined for the **GrayScale**, **PseudoColor**, and **DirectColor** visual types. For **StaticGray**, **StaticColor**, and **TrueColor** visual types, the entries have defined values, but those values are specific to the visual and are not defined by Enhanced X-Windows. For the **StaticGray**, **StaticColor**, and **TrueColor** visual types, the *Allocate* parameter must be the value of **AllocNone**. For the other visual types, if the *Allocate* parameter is the **AllocNone** value, the colormap has no initially allocated entries, and clients can allocate them.

If the *Allocate* parameter is the value of **AllocAll**, the entire colormap is allocated as writable. The initial values of all allocated entries are undefined. For the **GrayScale** and **PseudoColor** visual types, the effect is the same as if the **XAllocColorCells** subroutine returned all pixel values from 0 to N – 1, where N is the value of the *map_entries* field in the specified visual. For the **DirectColor** visual type, the effect is the same as if the **XAllocColorPlanes** subroutine returned a pixel value of 0 and the *RedMaskReturn, GreenMaskReturn,* and *BlueMaskReturn* parameter values containing the same bits as the corresponding masks in the specified visual. However, in all cases, none of these entries can be freed by using the **XFreeColors** subroutine.

Parameters

Allocate	Specifies the colormap entries to be allocated.
DisplayPtr	Specifies the connection to the X Server.
VisualPtr	Specifies a pointer to a visual type supported on the screen.
WindowID	Specifies the window ID for the screen where the colormap is to be created.

Error Codes

BadAlloc

BadImplementation

BadMatch

BadValue

BadWindow

Implementation Specifics

This Xlib subroutine is part of AIXwindows Run Time Environment in AIXwindows Environment/6000.

Related Information

The **CreateColormap** protocol request.

XCreateFontCursor Subroutine

Purpose

Creates a cursor from a standard font.

Libraries

Enhanced X-Windows Library (**libX11.a**)

FORTRAN 77 Library (**libXfx.a**)

C Syntax

#include <X11/cursorfont.h>

Cursor XCreateFontCursor(*DisplayPtr*, *Shape*)
**Display ****DisplayPtr*;
unsigned int *Shape*;

FORTRAN Syntax

integer*4 fxcreatefontcursor
external fxcreatefontcursor
integer*4 *DisplayPtr*
integer*4 *Shape*
integer*4 *Cursor*
Cursor **= fxcreatefontcursor(***DisplayPtr*, *Shape*)

Description

The **XCreateFontCursor** subroutine creates a cursor from a standard font. A set of standard cursor shapes is available in a special font named *cursor.snf*. The **cursorfont.h** file contains the definitions of each of the cursor shapes. This font can be customized for individual display types.

The *Shape* parameter specifies which glyph (image) of the standard fonts to use. The hotspot (the point in the cursor corresponding to the coordinates reported for the pointer) comes from the information stored in the cursor font. The initial colors of a cursor are a black foreground and a white background.

Parameters

DisplayPtr	Specifies the connection to the X Server.
Shape	Specifies the shape for the cursor.

Error Codes

BadAlloc

BadImplementation

BadMatch

BadValue

Implementation Specifics

This Xlib subroutine is part of AIXwindows Run Time Environment in AIXwindows Environment/6000.

Related Information

The **CreateGlyphCursor** protocol request.

XCreateGC Subroutine

Purpose

Creates a new graphics context.

Libraries

Enhanced X-Windows Library (**libX11.a**)

FORTRAN 77 Library (**libXfx.a**)

C Syntax

GC XCreateGC(*DisplayPtr*, *DrawableID*, *ValueMaskCreate*, *Values*)
Display **DisplayPtr*;
Drawable *DrawableID*;
unsigned long *ValueMaskCreate*;
XGCValues **Values*;

FORTRAN Syntax

integer*4 fxcreategc
external fxcreategc
integer*4 *DisplayPtr*
integer*4 *DrawableID*
integer*4 *ValueMaskCreate*
integer*4 *Values*
integer*4 *GraphicsContext*
GraphicsContext = **fxcreategc**(*DisplayPtr*, *DrawableID*, *ValueMaskCreate*, *Values*)

Description

The **XCreateGC** subroutine creates a new graphics context that can be used with any destination drawable with the same root and depth as the specified drawable.

Parameters

DisplayPtr	Specifies the connection to the X Server.
DrawableID	Specifies the drawable.
ValueMaskCreate	Specifies the parameters in the graphics context being created to be set using the information in the **XGCValues** data structure. This parameter is the bitwise inclusive OR of one or more of the valid values graphics context component mask bits.
Values	Specifies any values as specified by the *ValueMask* parameter.

Error Codes

BadAlloc

BadDrawable

BadFont

BadImplementation

BadMatch

BadPixmap

BadValue

Implementation Specifics

This Xlib subroutine is part of AIXwindows Run Time Environment in AIXwindows Environment/6000.

Related Information

The **XGCValues** data structure and valid components.

The **CreateGC** protocol request.

The **XFreeGC** subroutine, **XCopyGC** subroutine, **XChangeGC** subroutine.

XCreateGlyphCursor Subroutine

Purpose

Creates a cursor from font glyphs.

Libraries

Enhanced X-Windows Library (**libX11.a**)

FORTRAN 77 Library (**libXfx.a**)

C Syntax

Cursor XCreateGlyphCursor(*DisplayPtr, SourceFont, MaskFont, SourceCharacter, MaskCharacter, ForegroundColor, BackgroundColor***)**
**Display ****DisplayPtr*;
Font *SourceFont, MaskFont*;
unsigned int *SourceCharacter, MaskCharacter*;
**XColor ****ForegroundColor*;
**XColor ****BackgroundColor*;

FORTRAN Syntax

integer*4 fxcreateglyphcursor
external fxcreateglyphcursor
integer*4 *DisplayPtr*
integer*4 *SourceFont, MaskFont*
integer*4 *SourceCharacter, MaskCharacter*
integer*4 *ForegroundColor, BackgroundColor*
integer*4 *Cursor*
Cursor **= fxcreateglyphcursor(***DisplayPtr, SourceFont, MaskFont, SourceCharacter, MaskCharacter, ForegroundColor, BackgroundColor***)**

Description

The **XCreateGlyphCursor** subroutine creates a cursor from font glyphs (images). The source and mask bitmaps are obtained from the specified font glyphs. When using the **XCreateGlyphCursor** subroutine, note the following:

- The *SourceCharacter* parameter must be a defined glyph in the *SourceFont* parameter.
- If the *MaskFont* parameter is specified, the *MaskCharacter* parameter must be a defined glyph in the *MaskFont* parameter. The *MaskFont* parameter and *MaskCharacter* parameter are optional.
- If defined, the origins of the *SourceCharacter* parameter and *MaskCharacter* parameter glyphs are positioned coincidently and define the hotspot. The *SourceCharacter* parameter and the *MaskCharacter* parameter do not need to have the same bounding box metrics. Also, there is no restriction on the placement of the hotspot relative to the bounding boxes. If no *MaskCharacter* parameter is given, all pixels of the source are displayed.
- 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.

The user must use the **XFreeFont** subroutine to free the fonts if no further explicit references to them are made.

Parameters

BackgroundColor	Specifies the red, green, and blue (RGB) values for the background of the source.
DisplayPtr	Specifies the connection to the X Server.
ForegroundColor	Specifies the red, green, and blue (RGB) values for the foreground of the source.
MaskCharacter	Specifies the character glyph for the mask.
MaskFont	Specifies the font for the mask glyph, or the **None** value.
SourceCharacter	Specifies the character glyph for the source.
SourceFont	Specifies the font for the source glyph.

Error Codes

BadAlloc

BadFont

BadImplementation

BadValue

Implementation Specifics

This Xlib subroutine is part of AIXwindows Run Time Environment in AIXwindows Environment/6000.

Related Information

The **CreateGlyphCursor** protocol request.

XCreateImage Subroutine

Purpose

Allocates the memory for the **XImage** subroutine.

Libraries

Enhanced X-Windows Library (**libX.a**)

FORTRAN 77 Library (**libXfx.a**)

C Syntax

XImage *XCreateImage(*DisplayPrt, VisualPtr, Depth, FormatOffset, Data, Width, HeightBitmapPad, BytesPerLine***)**
Display **DisplayPtr*;
Visual **VisualPtr*;
unsigned int *Depth*;
int *Format*;
int *Offset*;
char **Data*;
unsigned int *Width*;
unsigned int *Height*;
int *BitmapPad*;
int *BytesPerLine*;

FORTRAN Syntax

integer*4 fxcreateimage
external fxcreateimage
integer*4 *DisplayPtr, VisualPtr, Depth, Format, Offset*
integer*4 *Data*
integer*4 *Width, Height, BitmapPad, Bytesperline*
integer*4 *Image*
Image = **fxcreateimage(***DisplayPtr, VisualPtr, Depth, Format, Offset, Data, Width, Height, Bitmappad, Bytesperline***)**

Description

The **XCreateImage** subroutine allocates the memory for an **XImage** data structure for a specified display device, but allocates no memory for the image itself. Rather, it initializes the byte-order, bit-order, and bitmap-unit values from the display device and returns a pointer to the **XImage** data structure.

- The red, green, and blue mask values, derived from the visual structure, are defined for Z format images only.
- The *Offset* parameter permits rapid displaying of the image without requiring each scanline to be shifted into position.
- The *BitmapPad* parameter specifies the quantum of a scanline as 8-, 16-, or 32-bits. Thus, the start of one scanline in client memory is separated from the start of the next scanline by an integer multiple of the number of bits indicated in this parameter.
- If the *BytesPerLine* parameter is set to a value of 0, scanlines in memory are displayed contiguously and the value of the *BytesPerLine* parameter is calculated by the **Xlib** library.

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 use to call these functions through the image object are defined in the **<X11/Xutil.h>** file. To use your own routines for **XGetPixel**, **XSetPixel**, **XSubImage**, and **XAddPixel**, change the definitions in **Xutil.h**.

When an image is created using the **XCreateImage** subroutine, the **XDestroyImage** subroutine frees both the image structure and the data pointed to by the image structure.

Parameters

BitmapPad	Specifies the quantum of a scanline.
BytesPerLine	Specifies the number of bytes in the client image between the start of one scanline and the start of the next.
Data	Specifies a pointer to the image data.
Depth	Specifies the depth of the image.
DisplayPtr	Specifies the connection to the X Server.
Format	Specifies a **XYBitmap, XYPixmap,** or **ZPixmap** format for the image.
Height	Specifies the height of the image, in pixels.
Offset	Specifies the number of pixels to ignore at the beginning of the scanline.
VisualPtr	Specifies a pointer to the visual.
Width	Specifies the width of the image, in pixels.

Error Code

BadImplementation

Implementation Specifics

This Xlib subroutine is part of AIXwindows Run Time Environment in AIXwindows Environment/6000.

Related Information

The **XImage** data structure.

The **XDestroyImage** subroutine, **XPutImage** subroutine.

XCreatePixmap Subroutine

Purpose

Creates a pixmap of the specified size.

Libraries

Enhanced X-Windows Library (**libX11.a**)

FORTRAN 77 Library (**libXfx.a**)

C Syntax

Pixmap XCreatePixmap(*DisplayPtr, DrawableID, Width, Height, Depth***)**
Display ****DisplayPtr*;**
Drawable *DrawableID*;
unsigned int *Width, Height*;
unsigned int *Depth*;

FORTRAN Syntax

integer*4 fxcreatepixmap
external fxcreatepixmap
integer*4 *DisplayPtr*
integer*4 *DrawableID*
integer*4 *Width, Height, Depth*
integer*4 *Pixmap*
Pixmap = **fxcreatepixmap(***DisplayPtr, DrawableID, Width, Height, Depth***)**

Description

The **XCreatePixmap** subroutine creates a pixmap of a specified size and assigns it a pixmap ID. This function can be used with an **InputOnly** window as the *Drawable* parameter. The *Width* and *Height* parameters must be nonzero, and the *Depth* parameter must be a value supported by the screen of the specified drawable.

Which screen receives the pixmap is determined by the *Drawable* parameter. The pixmap can be used only on this screen and only with other drawables of the same depth. (See the **XCopyPlane** subroutine for an exception to this rule.)

The initial contents of the pixmap are undefined.

Parameters

Depth	Specifies the depth of the pixmap.
DisplayPtr	Specifies the connection to the X Server.
DrawableID	Specifies the screen on which to create the pixmap.
Height	Specifies the height of the pixmap.
Width	Specifies the width of the pixmap.

Error Codes

BadAlloc

BadDrawable

BadImplementation

BadValue

Implementation Specifics

This Xlib subroutine is part of AIXwindows Run Time Environment in AIXwindows Environment/6000.

Related Information

The **CreatePixmap** protocol request.

The **XCopyPlane** subroutine.

XCreatePixmapCursor Subroutine

Purpose

Creates a cursor from a pixmap.

Libraries

Enhanced X-Windows Library (**libX11.a**)

FORTRAN 77 Library (**libXfx.a**)

C Syntax

Cursor XCreatePixmapCursor(*DisplayPtr*, *Source*, *Mask*, *ForegroundColor*, *BackgroundColor*, X, *Y*)
Display **DisplayPtr*;
Pixmap *Source*;
Pixmap *Mask*;
XColor **ForegroundColor*;
XColor **BackgroundColor*;
unsigned int *X*, *Y*;

FORTRAN Syntax

integer*4 fxcreatepixmapcursor
external fxcreatepixmapcursor
integer*4 *DisplayPtr*
integer*4 *Source*, *Mask*
integer*4 *ForegroundColor*, *BackgroundColor*
integer*4 *X*, *Y*
integer*4 *Cursor*
Cursor = **fxcreatepixmapcursor(***DisplayPtr*, *Source*, *Mask*, *ForegroundColor*, *BackgroundColor*, *X*, *Y*)

Description

The **XCreatePixmapCursor** subroutine creates a cursor and returns the cursor ID associated with it.

- The red, green, and blue (RGB) values for the *ForegroundColor* and *BackgroundColor* parameters must be specified, even if the X Server only has a **StaticGray** or **GrayScale** screen.
- The foreground color is used for pixels set to a value of **1** in the source, and the background color is used for the pixels set to a value of **0**.
- Both the *Source* and *Mask* parameters, if specified, can have any root drawable, but must have a depth value of **1**.
- The *Mask* parameter defines the shape of the cursor. The pixels set to a value of **1** in the mask define the source pixels to be displayed, while the pixels set to a value of **0** define which pixels are ignored. If no mask is given, all source pixels are displayed. The mask, if present, must be the same size as the pixmap defined by the *Source* parameter, and the hotspot must be a point within the source. Pixmaps are freed using the **XFreePixmap** subroutine.
- The components of the cursor can be transformed arbitrarily to meet display limitations. The pixmaps can be freed immediately when no further explicit references to them are to

be made. Subsequent drawing in the source or mask pixmap has an undefined effect on the cursor (The X Server might or might not make a copy of the pixmap).

Parameters

BackgroundColor	Specifies the red, green, and blue (RGB) values for the background of the source.
DisplayPtr	Specifies the connection to the X Server.
ForegroundColor	Specifies the red, green, and blue (RGB) values for the foreground of the source.
Mask	Specifies the source bits of the cursor that is to be displayed, or **None**.
Source	Specifies the shape of the source cursor.
X	Specifies the x coordinate to indicate the hotspot relative to the origin of the source.
Y	Specifies the y coordinate to indicate the hotspot relative to the origin of the source.

Error Codes

BadAlloc

BadImplementation

BadMatch

BadPixmap

Implementation Specifics

This Xlib subroutine is part of AIXwindows Run Time Environment in AIXwindows Environment/6000.

Related Information

The **CreateCursor** protocol request.

XCreatePixmapFromBitmapData Subroutine

Purpose

Creates a pixmap using the bitmap-formated data.

Libraries

Enhanced X-Windows Library (**libX11.a**)

FORTRAN 77 Library (**libXfx.a**)

C Syntax

Pixmap XCreatePixmapFromBitmapData(*DisplayPtr, DrawableID, Data, Width, Height, Foreground, Background, Depth***)**
Display ****DisplayPtr*;**
Drawable *DrawableID*;
char ****Data*;**
unsigned int *Width, Height*;
unsigned long *Foreground, Background*;
unsigned int *Depth*;

FORTRAN Syntax

integer*4 fxcreatepixmapfrombitmapdata
external fxcreatepixmapfrombitmapdata
integer*4 *DisplayPtr, DrawableID*
integer*4 *Data*
integer*4 *Width, Height*
integer*4 *Foreground, Background, Depth*
integer*4 *Pixmap*
Pixmap **= fxcreatepixmapfrombitmapdata(***DisplayPtr, DrawableID, Data, Width, Height, Foreground, Background, Depth***)**

Description

The **XCreatePixmapFromBitmapData** subroutine creates a pixmap of the specified depth. It then calls the **XPutImage** subroutine to store the data in the pixmap. The depth must be supported by the screen of the specified drawable.

Parameters

Background	Specifies the background pixel values.
Data	Specifies the data in bitmap format.
Depth	Specifies the depth of the pixmap.
DisplayPtr	Specifies the connection to the X Server.
DrawableID	Specifies the drawable that indicates the screen.
Foreground	Specifies the foreground pixel values.
Height	Specifies the height of the pixmap.
Width	Specifies the width of the pixmap.

Error Codes

BadAlloc

BadImplementation

BadMatch

Implementation Specifics

This Xlib subroutine is part of AIXwindows Run Time Environment in AIXwindows Environment/6000.

Related Information

The **XPutImage** subroutine.

XCreateRegion Subroutine

Purpose

Creates a new empty region.

Libraries

Enhanced X-Windows Library (**libX11.a**)

FORTRAN 77 Library (**libXfx.a**)

C Syntax

Region XCreateRegion( )

FORTRAN Syntax

integer*4 fxcreateregion
external fxcreateregion
integer*4 *NewRegion*
NewRegion = **fxcreateregion( )**

Description

The **XCreateRegion** subroutine creates a new empty region.

Implementation Specifics

This Xlib subroutine is part of AIXwindows Run Time Environment in AIXwindows Environment/6000.

XCreateSimpleWindow Subroutine

Purpose

Creates an unmapped **InputOutput** subwindow.

Libraries

Enhanced X-Windows Library (**libX11.a**)

FORTRAN 77 Library (**libXfx.a**)

C Syntax

Window XCreateSimpleWindow(*DisplayPtr, Parent, X, Y , Width, Height, BorderWidth, Border, Background***)**
**Display ****DisplayPtr*;
Window *Parent*;
int *X, Y*;
unsigned int *Width, Height, BorderWidth*;
unsigned long *Border*;
unsigned long *Background*;

FORTRAN Syntax

integer*4 fxcreatesimplewindow
external fxcreatesimplewindow
integer*4 *DisplayPtr*
integer*4 *Parent*
integer*4 *X, Y*
integer*4 *Width, Height, BorderWidth*
integer*4 *Border, Background*
integer*4 *WindowID*
WindowID = **fxcreatesimplewindow(***DisplayPtr, Parent, X, Y, Width, Height, BorderWidth, Border, Background***)**

Description

The **XCreateSimpleWindow** subroutine 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 sibling windows. Any part of the window that extends outside its parent window is clipped.

The created window inherits the depth, class, and visual attributes of the parent window. All other window attributes use the default values.

For the created window to be visible on the screen, the window and all of its ancestor windows must be mapped and it cannot be obscured by any of its ancestor windows. Then the created window can be displayed by calling the **XMapWindows** subroutine.

Initially, the created window has the same cursor as the parent window. The *Cursor* parameter for the created window has a value of **None**. Use the **XDefineCursor** subroutine to define a new cursor for the created window.

Parameters

Background	Specifies the pixel value to be set for the background of the created window.
Border	Specifies the border pixel of the created window.
BorderWidth	Specifies the width (in pixels) of the border of the created window. The *BorderWidth* parameter for an **InputOnly** window must be 0, or a **BadMatch** error results.
DisplayPtr	Specifies the connection to the X Server.
Height	Specifies the height, which is the dimension of the inside of the created window, excluding the border. This parameter must be nonzero, or a **BadValue** error results.
Parent	Specifies the parent window ID.
Width	Specifies the width, which is the dimension of the inside of the created window, excluding the border. This parameter must be nonzero, or a **BadValue** error results.
X	Specifies the x coordinate. This coordinate, which is relative to the inside of the border of the parent window, defines the top-left outside corner of the border of the created window.
Y	Specifies the y coordinate. This coordinate, which is relative to the inside of the border of the parent window, defines the top-left outside corner of the border of the created window.

Error Codes

BadAlloc

BadImplementation

BadMatch

BadValue

BadWindow

Implementation Specifics

This Xlib subroutine is part of AIXwindows Run Time Environment in AIXwindows Environment/6000.

Related Information

The **CreateWindow** protocol request.

The **XCreateWindow** subroutine.

XCreateWindow Subroutine

Purpose

Creates an unmapped subwindow.

Libraries

Enhanced X-Windows Library (**libX11.a**)

FORTRAN 77 Library (**libXfx.a**)

C Syntax

Window XCreateWindow(*DisplayPtr*, *Parent*, *X*, *Y*,*Width*, *Height*, *BorderWidth*, *Depth*, *Class*, *VisualPtr*, *ValueMask*, Attributes)
Display **DisplayPtr*;
Window *Parent*;
int *X*, *Y*;
unsigned int *Width*, *Height*;
unsigned int *BorderWidth*;
int *Depth*;
unsigned int *Class*;
Visual **VisualPtr*;
unsigned long *ValueMask*;
XSetWindowAttributes **Attributes*;

FORTRAN Syntax

integer*4 fxcreatewindow
external fxcreatewindow
integer*4 *DisplayPtr*
integer*4 *Parent*
integer*4 *X*, *Y*
integer*4 *Width*, *Height*, *BorderWidth*, *Depth*
integer*4 *Class*, *VisualPtr*, *ValueMask*, *Attributes*
integer*4 *WindowID*
WindowID = **fxcreatewindow**(*DisplayPtr*, *Parent*, *X*, *Y*, *Width*, *Height*, *BorderWidth*, *Depth*, *Class*, *VisualPtr*, *ValueMask*, *Attributes*)

Description

The **XCreateWindow** subroutine creates an unmapped 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 sibling windows.

For the created window to be visible on the screen, the window and all of its ancestor windows must be mapped and it cannot be obscured by any of its ancestor windows. The created window can then be displayed by using the **XMapWindow** subroutine.

Initially, the created window has the same cursor as the parent window. The cursor for the created window has a value of **None**. Use the **XDefineCursor** subroutine to define a new cursor for the created window.

For an **InputOutput** window, the visual type and depth must be a combination supported by the screen. The depth does not need to be the same as the parent window, but if the parent window is an **InputOnly** window, a **Bad Match** error will result.

For an **InputOnly** window, the depth and border width must be 0 and the visual must be supported by the screen. The parent window, however, can have any depth and class.

The only window attributes defined for **InputOnly** windows are the *win_gravity*, *event_mask*, *do_not_propagate_mask*, *override_redirect*, and *cursor* attributes. Any other attribute will result in a **BadMatch** error.

Parameters

Attributes — Specifies the window attributes to be set when the window is created. The *Valuemask* parameter should have the appropriate bits set to indicate which attributes in the structure were set.

Borderwidth — Specifies the width of the border of the new window in pixels. The *Borderwidth* parameter for an **InputOnly** window must be zero, or a **BadMatch** error results.

Class — Specifies the class of the created window. The *Class* parameter can be:

- **InputOutput** — Indicates a window that cannot be used for graphics requests.
- **InputOnly** — Indicates a normal type of window, which is used for both input and output.
- **CopyFromParent** — Indicates that the class is taken from the parent window.

Depth — Specifies the depth of the new window. If the *Depth* parameter is **CopyFromParent**, the *Depth* is taken from the parent window.

DisplayPtr — Specifies the connection to the X Server.

Height — Specifies the height dimension of the inside of the created window. This dimension does not include the border of the created window, which is entirely outside of the window. This parameter must be nonzero.

Parent — Specifies the parent window ID.

Valuemask — Specifies the window attributes to be set in the *Attributes* parameter. This mask is the bitwise-inclusive OR of the valid attribute mask bits. If the *Valuemask* parameter is 0, the attributes are ignored and not referenced.

VisualPtr — Specifies the visual type. If the *VisualPtr* parameter is **CopyFromParent**, the visual type is taken from the parent window.

Width — Specifies the width dimension of the inside of the created window. This dimension does not include the border of the created window, which is entirely outside of the window. This parameter must be nonzero.

X — Specifies the x coordinate, which defines the top-left, outside corner of the border relative to the inside of the borders of the parent window.

Y Specifies the y coordinate, which defines the top-left, outside corner of the border relative to the inside of the borders of the parent window.

Error Codes

BadAlloc

BadColor

BadCursor

BadImplementation

BadMatch

BadPixmap

BadValue

BadWindow

Implementation Specifics

This Xlib subroutine is part of AIXwindows Run Time Environment in AIXwindows Environment/6000.

Related Information

The **CreateWindow** protocol request.

The **XDefineCursor** subroutine, **XMapWindow** subroutine.

XDefineCursor Subroutine

Purpose

Defines a cursor for a window.

Libraries

Enhanced X-Windows Library (**libX11.a**)

FORTRAN 77 Library (**libXfx.a**)

C Syntax

XDefineCursor(*DisplayPtr, WindowID, CursorID)*
Display **DisplayPtr*;
Window *WindowID;*
Cursor *CursorID*;

FORTRAN Syntax

external fxdefinecursor
integer*4 *DisplayPtr*
integer*4 *WindowID, CursorID*
call fxdefinecursor(*DisplayPtr, WindowID, CursorID***)**

Description

The **XDefineCursor** subroutine defines which cursor will be used in a window.

Setting the *Cursor* parameter to a value of **None** is equivalent to using the **XUndefineCursor** subroutine.

Parameters

CursorPtr	Specifies the cursor to be displayed when the pointer is in the specified window. If no cursor is to be displayed, it has a value of **None**.
DisplayID	Specifies the connection to the X Server.
WindowID	Specifies the window ID.

Error Codes

BadAlloc

BadCursor

BadImplementation

BadWindow

Implementation Specifics

This Xlib subroutine is part of AIXwindows Run Time Environment in AIXwindows Environment/6000.

Related Information

The **ChangeWindowAttributes** protocol request.

The **XUndefineCursor** subroutine.

XDeleteAssoc Subroutine

Purpose

Deletes an entry from a specific associate table.

Library

Enhanced X-Windows Library (**liboldX.a**)

Syntax

```
#include <X11/X10.h>
XDeleteAssoc(DisplayPtr, Table, x_id)
 Display *DisplayPtr;
 XAssocTable *Table;
 XID x_id;
```

Description

The **XDeleteAssoc** subroutine deletes an entry from a specific associate table. It deletes an association in an **XAssocTable** structure 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** structure.

Note: This subroutine is in the **liboldX.a** library. Include this library in the compiler command to build your program. For example:

```
{compiler-option} -o samples samples.c -loldX -lX11
```

Parameters

DisplayPtr	Specifies the connection to the X Server.
Table	Specifies the associate table.
x_id	Specifies the XID.

Implementation Specifics

This Xlib subroutine is part of AIXwindows Run Time Environment in AIXwindows Environment/6000.

Related Information

The **XCreateAssocTable** subroutine.

XDeleteContext Subroutine

Purpose

Deletes data associated with a specified window and context type.

Libraries

Enhanced X-Windows Library (**libX11.a**)

FORTRAN 77 Library (**libXfx.a**)

C Syntax

int XDeleteContext(*DisplayPtr*, *WindowID*, *Context*)
**Display ****DisplayPtr*;
Window *WindowID*;
XContext *Context;*

FORTRAN Syntax

integer*4 fxdeletecontext
external fxdeletecontext
integer*4 *DisplayPtr*, *WindowID*, *Context*
integer*4 *Status*
Status = **fxdeletecontext(***DisplayPtr*, *WindowID*, *Context*)

Description

The **XDeleteContext** subroutine deletes from the data structure the entry and context type for a specified window.

The **XDeleteContext** subroutine does not free the data for the address that was saved.

Parameters

Context	Specifies the context type to which the data belongs.
DisplayPtr	Specifies the connection to the X Server.
WindowID	Specifies the window ID with which the data is associated.

Return Values

0	The **XDeleteContext** subroutine executes routinely.
Nonzero	The subroutine cannot execute.

Error Codes

BadImplementation

XCNOENT (context-not-found)

Implementation Specifics

This Xlib subroutine is part of AIXwindows Run Time Environment in AIXwindows Environment/6000.

XDeleteModifiermapEntry Subroutine

Purpose

Deletes an entry from the **XModifierKeymap** data structure.

Libraries

Enhanced X-Windows Library (**libX11.a**)

FORTRAN 77 Library (**libXfx.a**)

C Syntax

XModifierKeymap *XDeleteModifiermapEntry(*ModifierMap***,** *Keycode***,** *Modifier***)**
XModifierKeymap ***ModifierMap*;**
KeyCode *Keycode***;**
int *Modifier***;**

FORTRAN Syntax

integer*4 fxdeletemodifiermapentry
external fxdeletemodifiermapentry
integer*4 *ModifierMap*
integer*4 *Keycode*
integer*4 *Modifier*
integer*4 *Keymap*
Keymap **= fxdeletemodifiermapentry(***ModifierMap***,** *Keycode***,** *Modifier***)**

Description

The **XDeleteModifiermapEntry** subroutine deletes a specified keycode from the data set that controls the specified modifier. It returns a pointer to the resulting **XModifierKeymap** data structure.

Parameters

Keycode	Specifies the keycode to be deleted.
Modifier	Specifies the modifier.
ModifierMap	Specifies a pointer to the **XModifierKeymap** data structure.

Error Code

BadImplementation

Implementation Specifics

This Xlib subroutine is part of AIXwindows Run Time Environment in AIXwindows Environment/6000.

Related Information

The **XModifierKeymap** data structure.

XDeleteProperty Subroutine

Purpose

Deletes a property for a specified window.

Libraries

Enhanced X-Windows Library (**libX11.a**)

FORTRAN 77 Library (**libXfx.a**)

C Syntax

XDeleteProperty(*DisplayPtr*, *WindowID*, *Property*)
Display **DisplayPtr*;
Window *WindowID;*
Atom *Property*;

FORTRAN Syntax

external fxdeleteproperty
integer*4 *DisplayPtr*
integer*4 *WindowID*
integer*4 *Property*
call fxdeleteproperty(*DisplayPtr*, *WindowID*, *Property*)

Description

The **XDeleteProperty** subroutine deletes a property for a specified window when that property was defined for the specified window. In this case the X Server generates a **PropertyNotify** event on the window.

Parameters

DisplayPtr	Specifies the connection to the X Server.
Property	Specifies the property name.
WindowID	Specifies the window ID.

Error Codes

BadAtom

BadImplementation

BadWindow

Implementation Specifics

This Xlib subroutine is part of AIXwindows Run Time Environment in AIXwindows Environment/6000.

Related Information

The **PropertyNotify** event.

The **DeleteProperty** protocol request.

The **XChangeProperty** subroutine.

XDestroyAssocTable Subroutine

Purpose

Frees the memory associated with a specific associate table.

Library

Enhanced X-Windows Library (**liboldX.a**)

Syntax

```
#include <X11/X10.h>
XDestroyAssocTable(Table)
  XAssocTable *Table;
```

Description

The **XDestroyAssocTable** subroutine frees the memory associated with a specific associate table. Using an **XAssocTable** structure after it has been destroyed will have unpredictable and probably disastrous consequences.

Note: This subroutine is in the **liboldX.a** library. Include this library in the compiler command to build your program. For example:

```
{compiler-option} -o samples samples.c -loldX -lX11
```

Parameters

Table	Specifies the associate table.

Implementation Specifics

This Xlib subroutine is part of AIXwindows Run Time Environment in AIXwindows Environment/6000.

Related Information

The **XCreateAssocTable** subroutine.

XDestroyImage Subroutine

Purpose

Deallocates memory associated with the **XImage** data structure.

Libraries

Enhanced X-Windows Library (**libX11.a**)

FORTRAN 77 Library (**libXfx.a**)

C Syntax

int XDestroyImage(*XImagePtr***)**
XImage ****XImagePtr*;**

FORTRAN Syntax

external fxdestroyimage
integer*4 *XImagePtr*
call fxdestroyimage(*XImagePtr***)**

Description

The **XDestroyImage** subroutine deallocates memory previously allocated using the **XCreateImage** subroutine.

Parameter

XImagePtr Specifies a pointer to the image.

Error Code

BadImplementation

Implementation Specifics

This Xlib subroutine is part of AIXwindows Run Time Environment in AIXwindows Environment/6000.

Related Information

The **XImage** data structure.

The **XCreateImage** subroutine.

XDestroyRegion Subroutine

Purpose

Frees the storage associated with a specified region.

Libraries

Enhanced X-Windows Library (**libX11.a**)

FORTRAN 77 Library (**libXfx.a**)

C Syntax

XDestroyRegion(*RegionPtr*)
Region *RegionPtr*;

FORTRAN Syntax

external fxdestroyregion
integer*4 *RegionPtr*
call fxdestroyregion(*RegionPtr*)

Description

The **XDestroyRegion** subroutine deallocates storage associated with a specified region.

Parameter

RegionPtr Specifies the region.

Error Code

BadImplementation

Implementation Specifics

This Xlib subroutine is part of AIXwindows Run Time Environment in AIXwindows Environment/6000.

XDestroySubwindows Subroutine

Purpose

Destroys all subwindows of a specified window.

Libraries

Enhanced X-Windows Library (**libX11.a**)

FORTRAN 77 Library (**libXfx.a**)

C Syntax

XDestroySubwindows(*DisplayPtr, WindowID)*
Display **DisplayPtr;*
Window *WindowID;*

FORTRAN Syntax

external fxdestroysubwindows
integer*4 *DisplayPtr*
integer*4 *WindowID*
call fxdestroysubwindows(*DisplayPtr, WindowID*)

Description

The **XDestroySubwindows** subroutine destroys all inferior windows of a 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 the **Expose** events on the specified window. The subwindows should not be referenced again.

Parameters

DisplayPtr	Specifies the connection to the X Server.
WindowID	Specifies the window ID of the window to be destroyed.

Error Codes

BadWindow

Implementation Specifics

This Xlib subroutine is part of AIXwindows Run Time Environment in AIXwindows Environment/6000.

XDestroyWindow Subroutine

Purpose

Unmaps and destroys a specified window and all its subwindows.

Libraries

Enhanced X-Windows Library (**libX11.a**)

FORTRAN 77 Library (**libXfx.a**)

C Syntax

XDestroyWindow(*DisplayPtr, WindowID)*
Display **DisplayPtr*;
Window *WindowID;*

FORTRAN Syntax

external fxdestroywindow
integer*4 *DisplayPtr*
integer*4 *WindowID*
call fxdestroywindow(*DisplayPtr, WindowID***)**

Description

The **XDestroyWindow** subroutine destroys a 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 specified, no windows are destroyed.)

If the window specified is a mapped window, the **XDestroyWindow** subroutine automatically unmaps it and destroys all its inferior windows. 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 sibling windows and across subhierarchies is not otherwise constrained.

Destroying a mapped window generates the **Expose** events on other windows that were obscured by the window being destroyed.

Parameters

DisplayPtr	Specifies the connection to the X Server.
WindowID	Specifies the window ID of the window to be destroyed.

Error Codes

BadWindow

BadImplementation

Implementation Specifics

This Xlib subroutine is part of AIXwindows Run Time Environment in AIXwindows Environment/6000.

Related Information

The **DestroyNotify** event, **Expose** event.

The **DestroyWindow** protocol request.

XDisableAccessControl Subroutine

Purpose

Disables use of the access control list.

Libraries

Enhanced X-Windows Library (**libX11.a**)

FORTRAN 77 Library (**libXfx.a**)

C Syntax

XDisableAccessControl(*DisplayPtr***)**
Display ***DisplayPtr;

FORTRAN Syntax

external fxdisableaccesscontrol
integer*4 *DisplayPtr*
call fxdisableaccesscontrol(*DisplayPtr***)**

Description

The **XDisableAccessControl** subroutine disables use of the access control list at each connection setup. The client application must reside on the same host as the X Server or have the required permission in the initial authorization at connection setup.

Parameter

DisplayPtr Specifies the connection to the X Server.

Error Codes

BadAccess

BadImplementation

Implementation Specifics

This Xlib subroutine is part of AIXwindows Run Time Environment in AIXwindows Environment/6000.

Related Information

The **SetAccessControl** protocol request.

The **XEnableAccessControl** subroutine.

XDisplayKeycodes Subroutine

Purpose

Gets the legal keycodes for a display.

Libraries

Enhanced X-Windows Library (**libX11.a**)

FORTRAN 77 Library (**libXfx.a**)

C Syntax

XDisplayKeycodes(*DisplayPtr*, *MinimumKeycodesReturn*, *MaximumKeycodesReturn*)
Display **DisplayPtr*;
int **MinimumKeycodesReturn;*
int **MaximumKeycodesReturn*;

FORTRAN Syntax

external fxdisplaykeycodes
integer*4 *DisplayPtr*
integer*4 *MinimumKeycodesReturn*
integer*4 *MaximumKeycodesReturn*
call fxdisplaykeycodes(*DisplayPtr*, *MinimumKeycodesReturn*, *MaximumKeycodesReturn*)

Description

The **XDisplayKeycodes** subroutine returns the minimum and maximum number of keycodes supported by a specified display device. The minimum is never less than 8; the maximum is never more than 255. Not all key codes in this range require corresponding keys.

Parameters

DisplayPtr	Specifies the connection to the X Server.
MinimumKeycodesReturn	Returns the minimum number of keycodes.
MaximumKeycodesReturn	Returns the maximum number of keycodes.

Error Code

BadImplementation

Implementation Specifics

This Xlib subroutine is part of AIXwindows Run Time Environment in AIXwindows Environment/6000.

XDisplayMotionBufferSize Subroutine

Purpose

Returns the size of the motion buffer.

Libraries

Enhanced X-Windows Library (**libX11.a**)

FORTRAN 77 Library (**libXfx.a**)

C Syntax

unsigned long XDisplayMotionBufferSize(*DisplayPtr***)**
Display ***DisplayPtr;

FORTRAN Syntax

external fxdisplaymotionbuffersize
integer*4 *DisplayPtr*
integer*4 *Size*
size = fxdisplaymotionbuffersize(*DisplayPtr***)**

Description

The **XDisplayMotionBufferSize** subroutine returns the size of the motion buffer. The server retains the recent history of the pointer motion, and does so to a finer granularity than is reported by **MotionNotify** events. The **XGetMotionEvents** subroutine makes this history available.

Parameter

DisplayPtr Specifies the connection to the X Server.

Error Code

BadImplementation

Implementation Specifics

This Xlib subroutine is part of AIXwindows Run Time Environment in AIXwindows Environment/6000.

Related Information

The **MotionNotify** event.

The **XGetMotionEvents** subroutine.

XDisplayName Subroutine

Purpose

Reports an error when the requested display deivce does not exist.

Libraries

Enhanced X-Windows Library (**libX11.a**)

FORTRAN 77 Library (**libXfx.a**)

C Syntax

char *XDisplayName(*String***)**
char ***String;

FORTRAN Syntax

character*256 fxdisplayname
external fxdisplayname
character*256 *ReturnString*
character*256 *String*
ReturnString = **fxdisplayname(***String***)**

Description

The **XDisplayName** subroutine returns the name of the display device that the **XOpenDisplay** subroutine would attempt to use. If a **NULL** string is specified, it looks in the environment for the display device name.

When the requested display device does not exist, the **XDisplayName** subroutine reports an error to the user.

Parameter

String Specifies the character string.

Error Code

BadImplementation

Implementation Specifics

This Xlib subroutine is part of AIXwindows Run Time Environment in AIXwindows Environment/6000.

Related Information

The **XOpenDisplay** subroutine.

XDraw Subroutine

Purpose

Achieves the effects of the **XDraw**, **XDrawDashed**, and **XDrawPatterned** subroutines from the RT X-Windows, Version 1.1.

Library

Enhanced X-Windows Library (**liboldX.a**)

Syntax

#include <X11/X10.h>

Status **XDraw**(*DisplayPtr*, *Drawable*, *GraphicsContext*, *VerticesList*, *VerticesCount*)
Display **DisplayPtr*;
Drawable *DrawableID*;
GC *GraphicsContext*;
Vertex **VerticesList*;
int *VerticesCount*;

Description

The **XDraw** subroutine draws an arbitrary polygon or curve which is defined by the specified list of Vertices as specified in the *VerticesList* parameter. The points are connected by lines specified in the flags in the vertex structure.

Each Vertex, as defined in the **<X11/X10.h>** header file, is a structure with the following elements:

```
typedef struct _Vertex {

   short x, y;
   unsigned short flags;

} Vertex;
```

The fields of the **Vertex** data structure include the following:

x Specifies the x coordinate of the vertex.

y Specifies the y coordinate of the vertex. These coordinates can be:

- Relative to the upper-left inside corner of the drawable if the **VertexRelative** flag is 0.
- Relative to the previous vertex if the **VertexRelative** flag is one.

flags Specifies the relationship to the vertex. These flags, as defined in the **<X11/X10.h>** header file, can be one of the following values:

VertexRelative	0x0001	else absolute
VertexDontDraw	0x0002	else draw
VertexCurved	0x0004	else straight

VertexStartClosed	0x0008	else not
VertexEndClosed	0x0010	else not

- If the **VertexRelative** value is not set, the coordinates are absolute (relative to the drawable). The first vertex must be an absolute vertex.
- If the **VertexDontDraw** value is 1, 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 the **VertexCurved** value is 1, 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 the **VertexCurved** value to 1 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.
- The **VertexDontDraw** bits and the **VertexCurved** bits can be set to 1. 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 the **VertexStartClosed** value is 1, 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 the **VertexEndClosed** bit set to one. The points in between form a cycle to determine predecessor and successor vertices for the spline algorithm.

The **XDraw** subroutine 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 subroutine also uses the graphics context mode-dependent components: *foreground*, *background*, *tile*, *stipple*, *ts_x_origin*, *ts_y_origin*, *dash_offset*, and *dash_list*.

Note: This subroutine is in the **liboldX.a** library. Include this library in the compiler command to build your program. For example:

```
{compiler-option} -o samples samples.c -loldX -lX11
```

Parameters

DisplayPtr	Specifies the connection to the X Server.
DrawableID	Specifies the drawable.
GraphicsContext	Specifies the graphics context.
VerticesList	Specifies a pointer to the list of vertices which indicate what to draw.
VerticesCount	Specifies the number of vertices in the *VerticesList* parameter.

Implementation Specifics

This Xlib subroutine is part of AIXwindows Run Time Environment in AIXwindows Environment/6000.

Related Information

The **XDrawFilled** subroutine.

XDrawArc Subroutine

Purpose

Draws a single arc in a specified drawable.

Libraries

Enhanced X-Windows Library (**libX11.a**)

FORTRAN 77 Library (**libXfx.a**)

C Syntax

XDrawArc(*DisplayPtr, DrawableID, GraphicsContext, X, Y, Width, Height, Angle1 , Angle2*)
Display **DisplayPtr*;
Drawable *DrawableID;*
GC *GraphicsContext*;
int *X, Y*;
unsigned int *Width, Height*;
int *Angle1, Angle2*;

FORTRAN Syntax

external fxdrawarc
integer*4 *DisplayPtr*
integer*4 *DrawableID, GraphicsContext*
integer*4 *X, Y, Width, Height, Angle1, Angle2*
call fxdrawarc(*DisplayPtr, DrawableID, GraphicsContext, X, Y, Width, Height, Angle1, Angle2*)

Description

The **XDrawArc** subroutine draws a single circular or elliptical arc in the specified drawable.

Each arc is specified by a rectangle and two angles. The center of the circle or ellipse is the center of the rectangle; the major and minor axes are specified by the *Width* and *Height* parameters. Positive angles indicate counterclockwise motion, and negative angles clockwise motion. If the *Angle2* parameter is greater than 360 degrees, it is truncated to 360 degrees.

For an arc specified as [`X, Y, Width, Height, Angle1, Angle2`], the origin of the major and minor axes is at [`X + Width/2, Y + Height/2`], and the infinitely thin path describing the entire circle or ellipse intersects the horizontal axis at [`X, Y + Height/2`] and [`X + Width, Y + Height/2`] intersects the vertical axis at [`X + Width/2, Y`] and [`X + Width/2, Y + Height`]. These coordinates can be fractional and 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 the arc specified by the two infinitely thin paths consisting of all points whose perpendicular distance from the path of the circle or ellipse is equal to `lw/2` (which may be a fractional value). The cap-style and join-style are applied the same as for a line corresponding to a tangent of the circle or ellipse at the endpoint.

For an arc specified as [`X, Y, Width, Height, Angle1, Angle2`], 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) * Width/Height] + Adjust
```

The skewed-angle and normal-angle are expressed in radians in the range [0, 2pi] and where atan returns a value in the range [–pi/2, pi/2] and `Adjust` is:

0	for normal-angle in the range [0, pi/2]
pi	for normal-angle in the range [pi/2, 3pi/2]
2pi	for normal-angle in the range [3pi/2, 2pi]

When two arcs intersect, the **XDrawArc** subroutine does not draw a pixel more than once if they join correctly and the line-width is greater than 0. Otherwise, intersecting pixels are drawn multiple times.

Specifying an arc with a given endpoint and a clockwise extent draws the same pixels as specifying the other endpoint and an equivalent counterclockwise extent, except as it affects intersecting arcs.

If the last point in an arc coincides with the first point in the next arc, the two arcs will intersect correctly. If the first point in the first arc coincides with the last point in the last arc, the two arcs will intersect correctly.

By specifying one axis to be 0, a horizontal or vertical line can be drawn. Angles are computed based solely on the coordinate system, and ignore aspect ratio.

The **XDrawArc** subroutine uses the *function, plane_mask, line_width, line_style, cap_style, join_style, fill_style, subwindow_mode, clip_x_origin, clip_y_origin*, and *clip_mask* graphics context fields. It also uses the *foreground, background, tile, stipple, ts_x_origin, ts_y_origin, dash_offset*, and *dash_list* graphics context mode-dependent fields.

Parameters

Angle1	Specifies the start of the arc relative to the three 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, not to exceed 360 degrees.
DisplayPtr	Specifies the connection to the X Server.
DrawableID	Specifies the drawable.
GraphicsContext	Specifies the graphics context.
Height	Specifies the height of the arc, which is its minor axes.
Width	Specifies the width of the arc, which is its major axes.
X	Specifies the x coordinate.
Y	Specifies the y coordinate.

Error Codes

BadDrawable

BadGC

BadImplementation

BadMatch

Implementation Specifics

This Xlib subroutine is part of AIXwindows Run Time Environment in AIXwindows Environment/6000.

Related Information

The **PolyArc** protocol request.

The **XDrawArcs** subroutine.

XDrawArcs Subroutine

Purpose

Draws multiple arcs in a specified drawable.

Libraries

Enhanced X-Windows Library (**libX11.a**)

FORTRAN 77 Library (**libXfx.a**)

C Syntax

XDrawArcs(*DisplayPtr*, *DrawableID*, *GraphicsContext*, *Arcs*, *NumberArcs*)
Display **DisplayPtr;*
Drawable *DrawableID*;
GC *GraphicsContext*;
XArc **Arcs*;
int *NumberArcs*;

FORTRAN Syntax

external fxdrawarcs
integer*4 *DisplayPtr*
integer*4 *DrawableID*, *GraphicsContext*
integer*4 *Arcs*, *NumberArcs*
call fxdrawarcs(*DisplayPtr*, *DrawableID*, *GraphicsContext*, *Arcs*, *NumberArcs*)

Description

The **XDrawArcs** subroutine draws multiple circular or elliptical arcs in the specified drawable.

Each arc is specified by a rectangle and two angles. The center of each circle or ellipse is the center of the rectangle; the major and minor axes are specified by the *Width* and *Height* parameters. Positive angles indicate counterclockwise motion, and negative angles clockwise motion. If the *Angle2* parameter is greater than 360 degrees, it is truncated to 360 degrees.

When two arcs intersect, the **XDrawArcs** subroutine does not draw a pixel more than once if they join correctly and the line-width is greater than 0. Otherwise, intersecting pixels are drawn multiple times.

Specifying an arc with a given endpoint and a clockwise extent draws the same pixels as specifying the other endpoint and an equivalent counterclockwise extent, except as it affects intersecting arcs.

If the last point in an arc coincides with the first point in the next arc, the two arcs will intersect correctly.

By specifying one axis to be 0, a horizontal or vertical line can be drawn. Angles are computed based solely on the coordinate system, and ignore aspect ratio.

The **XDrawArcs** subroutine uses the *function, plane_mask, line_width, line_style, cap_style, join_style, fill_style, subwindow_mode, clip_x_origin, clip_y_origin*, and *clip_mask* graphics context fields. It also uses the *foreground, background, tile, stipple, ts_x_origin, ts_y_origin, dash_offset*, and *dash_list* graphics context mode-dependent fields.

Parameters

Arcs	Specifies a pointer to an array of arcs.
DisplayPtr	Specifies the connection to the X Server.
DrawableID	Specifies the drawable.
GraphicsContext	Specifies the graphics context.
NumberArcs	Specifies the number of arcs in the array.

Error Codes

BadDrawable

BadGC

BadImplementation

BadMatch

Implementation Specifics

This Xlib subroutine is part of AIXwindows Run Time Environment in AIXwindows Environment/6000.

Related Information

The **PolyArc** protocol request.

The **XDrawArc** subroutine.

XDrawFilled Subroutine

Purpose

Draws polygons and curves and fills them.

Library

Enhanced X-Windows Library (**liboldX.a**)

Syntax

```
#include <X11/X10.h>

Status XDrawFilled(DisplayPtr, Drawable, GraphicsContext, VerticesList
                   , VerticesCount)
Display *DisplayPtr;
Drawable DrawableID;
GC GraphicsContext;
Vertex *VerticesList;
int VerticesCount;
```

Description

The **XDraw** subroutine draws arbitrary polygons or curves and fills them. The **XDraw** subroutine 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 subroutine also uses the graphics context mode-dependent components: *foreground*, *background*, *tile*, *stipple*, *ts_x_origin*, *ts_y_origin*, *dash_offset*, and *dash_list*.

Note: This subroutine is in the **liboldX.a** library. Include this library in the compiler command to build your program. For example:

```
{compiler-option} -o samples samples.c -loldX -lX11
```

Parameters

DisplayPtr	Specifies the connection to the X Server.
DrawableID	Specifies the drawable.
GraphicsContext	Specifies the graphics context.
VerticesList	Specifies a pointer to the list of vertices which indicate what to draw.
VerticesCount	Specifies the number of vertices in the *VerticesList* parameter.

Implementation Specifics

This Xlib subroutine is part of AIXwindows Run Time Environment in AIXwindows Environment/6000.

Related Information

The **XDraw** subroutine.

XDrawImageString Subroutine

Purpose

Draws 8-bit image text in a specified drawable.

Libraries

Enhanced X-Windows Library (**libX11.a**)

FORTRAN 77 Library (**libXfx.a**)

C Syntax

XDrawImageString(*DisplayPtr, DrawableID, GraphicsContext, X, Y, String, Length)*
Display **DisplayPtr*;
Drawable *DrawableID;*
GC *GraphicsContext*;
int *X, Y*;
char **String;*
int *Length*;

FORTRAN Syntax

external fxdrawimagestring
integer*4 *DisplayPtr*
integer*4 *DrawableID, GraphicsContext*
integer*4 *X, Y*
character*256 *String*
integer*4 *Length*
call fxdrawimagestring(*DisplayPtr, DrawableID, GraphicsContext, X, Y, String, Length*)

Description

The **XDrawImageString** subroutine draws 8-bit image text characters in the specified drawable. Some applications, particularly terminal emulators, need to print image text in which both the foreground and background bits of each character are painted. This subroutine draws both the foreground and background bits of each character.

Using both the foreground and background pixels of the graphics context, the **XDrawImageString** subroutine first fills a destination rectangle with the background pixel, then paints the text with the foreground pixel. The upper-left corner of the filled rectangle is at:

```
[X, Y - FontAscent]
```

The width is:

```
OverallWidth
```

The height is:

```
FontAscent + FontDescent
```

The *OverallWidth*, *FontAscent* and *FontDescent* values are equivalent to those returned by the **XQueryTextExtents** subroutine with the *GraphicsContext* and *String* parameters.

For fonts defined with 2-byte matrix indexing, each byte is used as a Byte2 field with a Byte1 field having a value of 0.

The **XDrawImageString** subroutine uses the *plane_mask, foreground, background, font, subwindow_mode, clip_x_origin, clip_y_origin*, and *clip_mask* graphics context fields.

Parameters

DisplayPtr	Specifies the connection to the X Server.
DrawableID	Specifies the drawable.
GraphicsContext	Specifies the graphics context.
Length	Specifies the number of characters in the string argument.
String	Specifies the character string.
X	Specifies the x coordinate, relative to the origin of the specified drawable, and defining the origin of the first character.
Y	Specifies the y coordinate, relative to the origin of the specified drawable, and defining the origin of the first character.

Error Codes

BadDrawable

BadGC

BadImplementation

BadMatch

Implementation Specifics

This Xlib subroutine is part of AIXwindows Run Time Environment in AIXwindows Environment/6000.

Related Information

The **ImageText8** protocol request.

The **XQueryTextExtents** subroutine.

XDrawImageString16 Subroutine

Purpose

Draws 2-byte image text in a specified drawable.

Library

Enhanced X-Windows Library (**libX11.a**)

FORTRAN 77 Library (**libXfx.a**)

C Syntax

XDrawImageString16(*DisplayPtr*, *DrawableID*, *GraphicsContext*, *X*, *Y*, *String*, *Length*)
Display **DisplayPtr*;
Drawable *DrawableID;*
GC *GraphicsContext*;
int *X*, *Y;*
XChar2b **String;*
int *Length*;

FORTRAN Syntax

external fxdrawimagestring16
integer*4 *DisplayPtr*
integer*4 *DrawableID*, *GraphicsContext*
integer*4 *X*, *Y*, *String*, *Length*
call fxdrawimagestring16(*DisplayPtr*, *DrawableID*, *GraphicsContext*, *X*, *Y*, *String*, *Length*)

Description

The **XDrawImageString16** subroutine draws 2-byte or 16-bit image text characters in the drawable specified. Some applications, particularly terminal emulators, need to print image text in which both the foreground and background bits of each character are painted. Fonts with 2-byte matrix indexing can be used in this subroutine.

Using both the foreground and background pixels of the graphics context, the **XDrawImageString16** subroutine first fills a destination rectangle with the background pixel, then paints the text with the foreground pixel The upper-left corner of the filled rectangle is at:.

```
[X, Y – FontAscent]
```

The width is:

```
OverallWidth
```

The height is:

```
FontAscent + FontDescent
```

The values for the *OverallWidth*, *FontAscent* and *FontDescent* parameters are equivalent to those returned using the **XQueryTextExtents** subroutine.

The *String* parameter for the **XDrawImageString16** subroutine is of the **XChar2B** type. The X Server interprets each **XChar2B** type as a 16-bit number that has been transmitted with the most-significant byte first. The *Byte1* parameter of the **XChar2B** structure is taken as the most-significant byte.

The **XDrawImageString16** subroutine uses the *plane_mask, foreground, background, font, subwindow_mode, clip_x_origin, clip_y_origin*, and *clip_mask* graphics context fields.

Parameters

DisplayPtr	Specifies the connection to the X Server.
DrawableID	Specifies the drawable.
GraphicsContext	Specifies the graphics context.
Length	Specifies the number of characters in the string argument.
String	Specifies the character string.
X	Specifies the x coordinates, relative to the origin of the specified drawable, and defining the origin of the first character.
Y	Specifies the y coordinate, relative to the origin of the specified drawable, and defining the origin of the first character.

Error Codes

BadDrawable

BadFont

BadGC

BadImplementation

BadMatch

Implementation Specifics

This Xlib subroutine is part of AIXwindows Run Time Environment in AIXwindows Environment/6000.

Related Information

The **ImageText16** protocol request, **PolyText16** protocol request.

The **XQueryTextExtents** subroutine.

XDrawLine Subroutine

Purpose

Draws a single line between two points in a drawable.

Libraries

Enhanced X-Windows Library (**libX11.a**)

FORTRAN 77 Library (**libXfx.a**)

C Syntax

XDrawLine(*DisplayPtr*, *DrawableID*, *GraphicsContext*, *X1*, *Y1*, *X2*, *Y2*)
Display **DisplayPtr*;
Drawable *DrawableID;*
GC *GraphicsContext*;
int *X1*, *Y1*, *X2*, *Y2*;

FORTRAN Syntax

external fxdrawline
integer*4 *DisplayPtr*
integer*4 *DrawableID*, *GraphicsContext*
integer*4 *X1*, *Y1*, *X2*, *Y2*
call fxdrawline(*DisplayPtr*, *DrawableID*, *GraphicsContext*, *X1*, *Y1*, *X2*, *Y2*)

Description

The **XDrawLine** subroutine uses the components of the specified graphics context to draw a line connecting the point defined by the *X1*, *Y1* perameters to the point defined by the *X2*, *Y2* parameters. It does not draw any pixel more than once. If lines intersect, the intersecting pixels are drawn multiple times. No joining is performed at coincident end points.

The **XDrawLine** subroutine uses the *function, plane_mask, line_width, line_style, cap_style, fill_style, subwindow_mode, clip_x_origin, clip_y_origin*, and *clip_mask* graphics context fields. It also uses the *foreground, background, tile, stipple, ts_x_origin, ts_y_origin, dash_offset*, and *dash_list* graphics context mode–dependent fields.

Parameters

DisplayPtr	Specifies the connection to the X Server.
DrawableID	Specifies the drawable.
GraphicsContext	Specifies the graphics context.
X1	Specifies the x coordinate points used to connect the line.
X2	Specifies the x coordinate points used to connect the line.
Y1	Specifies the y coordinate points used to connect the line.
Y2	Specifies the y coordinate points used to connect the line.

Error Codes

BadDrawable

BadGC

BadImplementation

BadMatch

Implementation Specifics

This Xlib subroutine is part of AIXwindows Run Time Environment in AIXwindows Environment/6000.

Related Information

The **PolySegment** protocol request.

The **XDrawLines** subroutine, **XDrawSegments** subroutine.

XDrawLines Subroutine

Purpose

Draws multiple lines in a specified drawable.

Libraries

Enhanced X-Windows Library (**libX11.a**)

FORTRAN 77 Library (**libXfx.a**)

C Syntax

XDrawLines(*DisplayPtr*, *DrawableID*, *GraphicsContext*, *Points*, *NumberPoints*, *Mode*)
Display **DisplayPtr*;
Drawable *DrawableID;*
GC *GraphicsContext*;
XPoint **Points;*
int *NumberPoints*;
int *Mode*;

FORTRAN Syntax

external fxdrawlines
integer*4 *DisplayPtr*
integer*4 *DrawableID*, *GraphicsContext*
integer*4 *Points*, *NumberPoints*, *Mode*
call fxdrawlines(*DisplayPtr*, *DrawableID*, *GraphicsContext*, *Points*, *NumberPoints*, *Mode*)

Description

The **XDrawLines** subroutine draws multiple lines in the specified drawable with the components of the specified graphics context. It uses these components to draw the `npoints-1` lines between each pair of points `(point[i], point[i+1])` in the array of **XPoint** data structures.

The **XDrawLines** subroutine draws the lines in the order listed in the array. The lines join correctly at all intermediate points. 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 (0 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** protocol request were a single filled shape.

CoordModeOrigin value treats all coordinates as relative to the point of origin. **CoordModePrevious** vlaue treats all coordinates, after the first, as relative to the previous point.

The **XDrawLines** subroutine uses the *function, plane_mask, line_width, line_style, cap_style, fill_style, subwindow_mode, clip_x_origin, clip_y_origin, clip_mask* and *join_style* graphics context fields. It also uses the *foreground, background, tile, stipple, ts_x_origin, ts_y_origin, dash_offset*, and *dash_list* graphics context mode-dependent fields.

Parameters

DisplayPtr	Specifies the connection to the X Server.
DrawableID	Specifies the drawable.
GraphicsContext	Specifies the graphics context.
Mode	Specifies the coordinate mode as either the **CoordModeOrigin** or **CoordModePrevious** value.
NumberPoints	Specifies the number of points in the array.
Points	Specifies a pointer to an array of points.

Error Codes

BadDrawable

BadGC

BadImplementation

BadMatch

BadValue

Implementation Specifics

This Xlib subroutine is part of AIXwindows Run Time Environment in AIXwindows Environment/6000.

Related Information

The **PolyLine** protocol request.

The **XDrawLine** subroutine, **XDrawSegments** subroutine.

XDrawPoint Subroutine

Purpose

Draws a single point in a specified drawable.

Libraries

Enhanced X-Windows Library (**libX11.a**)

FORTRAN 77 Library (**libXfx.a**)

C Syntax

XDrawPoint(*DisplayPtr*, *DrawableID*, *GraphicsContext*, *X*, *Y*)
Display **DisplayPtr*;
Drawable *DrawableID*;
GC *GraphicsContext*;
int *X*, *Y*;

FORTRAN Syntax

external fxdrawpoint
integer*4 *DisplayPtr*
integer*4 *DrawableID*
integer*4 *GraphicsContext*
integer*4 *X*, *Y*
call fxdrawpoint(*DisplayPtr*, *DrawableID*, *GraphicsContext*, *X*, *Y*)

Description

The **XDrawPoint** subroutine uses the foreground pixel and function components of the graphics context to draw a single point in the specified drawable.

The **XDrawPoint** subroutine uses the *function, plane_mask, foreground, subwindow_mode, clip_x_origin, clip_y_origin*, and *clip_mask* graphics context fields.

Parameters

DisplayPtr	Specifies the connection to the X server.
DrawableID	Specifies the drawable.
GraphicsContext	Specifies the graphics context.
X	Specifies the x coordinate where the point will be drawn.
Y	Specifies the y coordinate where the point will be drawn.

Error Codes

BadDrawable

BadGC

BadImplementation

BadMatch

Implementation Specifics

This Xlib subroutine is part of AIXwindows Run Time Environment in AIXwindows Environment/6000.

Related Information

The **PolyPoint** protocol request.

The **XDrawPoints** subroutine.

XDrawPoints Subroutine

Purpose

Draws multiple points in specified drawable.

Libraries

Enhanced X-Windows Library (**libX11.a**)

FORTRAN 77 Library (**libXfx.a**)

C Syntax

XDrawPoints(*DisplayPtr*, *DrawableID*, *GraphicsContext*, *Points*, *NumberPoints*, *Mode*)
Display **DisplayPtr*;
Drawable *DrawableID;*
GC *GraphicsContext;*
XPoint **Points;*
int *NumberPoints*;
int *Mode*;

FORTRAN Syntax

external fxdrawpoints
integer*4 *DisplayPtr*
integer*4 *DrawableID*, *Graphics*, *Points*
integer*4 *NumberPoints*, *Mode*
call fxdrawpoints(*DisplayPtr*, *DrawableID*, *GraphicsContext*, *Points*,*NumberPoints*, *Mode*)

Description

The **XDrawPoints** subroutine uses the foreground pixel and graphics context function components to draw multiple points in a specified drawable. It uses the values from the **XPoint** data structure in the order listed in the array.

The **CoordModeOrigin** value treats all coordinates as relative to the point of origin. The **CoordModePrevious** value treats all coordinates, after the first, as relative to the previous point.

The **XDrawPoints** subroutine uses the *function, plane_mask, foreground, subwindow_mode, clip_x_origin, clip_y_origin*, and *clip_mask* graphics context fields.

Parameters

DisplayPtr	Specifies the connection to the X server.
DrawableID	Specifies the drawable.
GraphicsContext	Specifies the graphics context.
Mode	Specifies the coordinate mode as either the **CoordModeOrigin** or the **CoordModePrevious** value.
NumberPoints	Specifies the number of points in the array.
Points	Specifies a pointer to an array of points.

Error Codes

BadDrawable

BadGC

BadImplementation

BadMatch

BadValue

Implementation Specifics

This Xlib subroutine is part of AIXwindows Run Time Environment in AIXwindows Environment/6000.

Related Information

The **PolyPoint** protocol request.

The**XDrawPoint** subroutine.

XDrawRectangle Subroutine

Purpose

Draws the outline of a single rectangle in a drawable.

Libraries

Enhanced X-Windows Library (**libX11.a**)

FORTRAN 77 Library (**libXfx.a**)

C Syntax

XDrawRectangle(*DisplayPtr*, *DrawableID*, *GraphicsContext*, *X*, *Y*, *Width* , *Height*)
Display **DisplayPtr*;
Drawable *DrawableID*;
GC *GraphicsContext*;
int *X*, *Y*;
unsigned int *Width*, *Height*;

FORTRAN Syntax

external fxdrawrectangle
integer*4 *DisplayPtr*
integer*4 *DrawableID*, *GraphicsContext*
integer*4 *X*, *Y*, *Width*, *Height*
call fxdrawrectangle(*DisplayPtr*, *DrawableID*, *GraphicsContext*, *X*, *Y*, *Width*, *Height*)

Description

The **XDrawRectangle** subroutine draws the outline of a single rectangle as if a five-point **PolyLine** protocol request were specified as shown in the following example:

```
[X,Y] [X+Width,Y] [X+Width,Y+Height] [X,Y+Height] [X,Y]
```

No pixel is drawn more than once for the specified rectangle.

The **XDrawRectangle** subroutine uses the *function, plane_mask, line_width, line_style, join_style, fill_style, subwindow_mode, clip_x_origin, clip_y_origin*, and *clip_mask* graphics context fields. It also uses the *foreground, background, tile, stipple, ts_x_origin, ts_y_origin, dash_offset*, and *dash_list* graphics context mode-dependent fields.

Parameters

DisplayPtr	Specifies the connection to the X Server.
DrawableID	Specifies the drawable.
GraphicsContext	Specifies the graphics context.
Height	Specifies the height dimensions of the rectangle.
Width	Specifies the width dimensions of the rectangle.
X	Specifies the x coordinate which defines the upper-left corner of the rectangle.
Y	Specifies the y coordinate which defines the upper-left corner of the rectangle.

Error Codes

BadDrawable

BadGC

BadImplementation

BadMatch

Implementation Specifics

This Xlib subroutine is part of AIXwindows Run Time Environment in AIXwindows Environment/6000.

Related Information

The **PolyLine** protocol request, **PolyRectangle** protocol request.

The **XDrawRectangles** subroutine.

XDrawRectangles Subroutine

Purpose

Draws outline of multiple rectangles in drawable.

Libraries

Enhanced X-Windows Library (**libX11.a**)

FORTRAN 77 Library (**libXfx.a**)

C Syntax

XDrawRectangles(*DisplayPtr*, *DrawableID*, *GraphicsContext*,*Rectangles*, *NumberRectangles*)
Display **DisplayPtr*;
Drawable *DrawableID*;
GC *GraphicsContext*;
XRectangle *Rectangles*[];
int *NumberRectangles*;

FORTRAN Syntax

external fxdrawrectangles
integer*4 *DisplayPtr*
integer*4 *DrawableID*, *GraphicsContext*
integer*4 *Rectangles*, *NumberRectangles*
call fxdrawrectangles(*DisplayPtr*, *DrawableID*, *GraphicsContext*, *Rectangles*, *NumberRectangles*)

Description

The **XDrawRectangle** subroutine draws the outline of multiple rectangles in the specified drawable as if a five-point **PolyLine** protocol request were specified for each rectangle as shown in the following example:

```
[X,Y] [X+Width,Y] [X+Width,Y+Height] [X,Y+Height] [X,Y]
```

No pixel is drawn more than once for the specified rectangles. If the rectangles intersect, the intersecting pixels are drawn multiple times.

The **XDrawRectangles** subroutine uses the *function, plane_mask, line_width, line_style, join_style, fill_style, subwindow_mode, clip_x_origin, clip_y_origin*, and *clip_mask* graphics context fields. It also uses the *foreground, background, tile, stipple, ts_x_origin, ts_y_origin, dash_offset*, and *dash_list* graphics context mode-dependent fields.

Parameters

DisplayPtr	Specifies the connection to the X Server.
DrawableID	Specifies the drawable.
GraphicsContext	Specifies the graphics context.
NumberRectangles	Specifies the number of rectangles in the array.
Rectangles	Specifies a pointer to an array of rectangles.

Error Codes

BadDrawable

BadGC

BadImplementation

BadMatch

Implementation Specifics

This Xlib subroutine is part of AIXwindows Run Time Environment in AIXwindows Environment/6000.

Related Information

The **PolyLine** protocol request, **PolyRectangle** protocol request.

The **XDrawRectangle** subroutine.

XDrawSegments Subroutine

Purpose

Draws multiple line segments in a specified drawable.

Libraries

Enhanced X-Windows Library (**libX11.a**)

FORTRAN 77 Library (**libXfx.a**)

C Syntax

XDrawSegments(*DisplayPtr, DrawableID, GraphicsContext, Segments, NumberSegments***)**
Display **DisplayPtr*;
Drawable *DrawableID*;
GC *GraphicsContext*;
XSegment **Segments*;
int *NumberSegments*;

FORTRAN Syntax

external fxdrawsegments
integer*4 *DisplayPtr*
integer*4 *DrawableID, GraphicsContext*
integer*4 *Segments, NumberSegments*
call fxdrawsegments(*DisplayPtr, DrawableID, GraphicsContext, Segments, NumberSegments***)**

Description

The **XDrawSegments** subroutine draws multiple, unconnected, lines. For each segment it draws a line connecting the point defined by the *X1, Y1* parameters and the point defined by the *X2, Y2* parameters in the order listed in the array of the **XSegment** data structure.

The **XDrawSegments** subroutine does not perform joining at coincident end points. No pixel is drawn more than once. If lines intersect, the intersecting pixels are drawn multiple times.

The **XDrawSegments** subroutine uses the *function, plane_mask, line_width, line_style, cap_style, fill_style, subwindow_mode, clip_x_origin, clip_y_origin*, and *clip_mask* graphics context fields. It also uses the *foreground, background, tile, stipple, ts_x_origin, ts_y_origin, dash_offset*, and *dash_list* graphics context mode-dependent fields.

Parameters

DisplayPtr	Specifies the connection to the X Server.
DrawableID	Specifies the drawable.
GraphicsContext	Specifies the graphics context.
NumberSegments	Specifies the number of segments in the array.
Segments	Specifies a pointer to an array of segments.

Error Codes

BadDrawable

BadGC

BadImplementation

BadMatch

Implementation Specifics

This Xlib subroutine is part of AIXwindows Run Time Environment in AIXwindows Environment/6000.

Related Information

The **PolySegment** protocol request.

The **XDrawLine** subroutine, **XDrawLines** subroutine.

XDrawString Subroutine

Purpose

Draws 8-bit characters in a specified drawable.

Library

Enhanced X-Windows Library (**libX11.a**)

FORTRAN 77 Library (**libXfx.a**)

C Syntax

XDrawString(*DisplayPtr*, *DrawableID*, *GraphicsContext*, *X*, *Y*,*String*, *Length*)
Display **DisplayPtr*;
Drawable *DrawableID*;
GC *GraphicsContext*;
int *X*, *Y*;
char **String*;
int *Length*;

FORTRAN Syntax

external fxdrawstring
integer*4 *DisplayPtr*
integer*4 *DrawableID*, *GraphicsContext*
integer*4 *X*, *Y*
character*256 *String*
integer*4 *Length*
call fxdrawstring(*DisplayPtr*, *DrawableID*, *GraphicsContext*, *X*, *Y*, *String*, *Length*)

Description

The **XDrawString** subroutine draws 8-bit text characters in a specified drawable. Each character image, as defined by the font in the graphics context, is treated as an additional mask for a fill operation on the drawable. The drawable is modified only where the font character has a bit set to 1.

The **XDrawString** subroutine uses the *function, plane_mask, fill_style, font, subwindow_mode, clip_x_origin, clip_y_origin*, and *clip_mask* graphics context fields. It also uses the *foreground, background, tile, stipple, ts_s_origin*, and *ts_y_origin* graphics context mode-dependent fields.

Parameters

DisplayPtr	Specifies the connection to the X Server.
DrawableID	Specifies the drawable.
GraphicsContext	Specifies the graphics context.
Length	Specifies the number of characters in the string parameter.
String	Specifies the character string.
X	Specifies the x coordinate relative to the origin of the specified drawable. This coordinates defines the baseline starting position for the initial character.

Y	Specifies the y coordinate relative to the origin of the specified drawable. This coordinate defines the baseline starting position for the initial character.

Error Codes

BadDrawable

BadFont

BadGC

BadImplementation

BadMatch

Implementation Specifics

This Xlib subroutine is part of AIXwindows Run Time Environment in AIXwindows Environment/6000.

Related Information

The **PolyText8** protocol request.

The **XDrawString16** subroutine.

XDrawString16 Subroutine

Purpose

Draws 2-byte characters in a specified drawable.

Libraries

Enhanced X-Windows Library (**libX11.a**)

FORTRAN 77 Library (**libXfx.a**)

C Syntax

XDrawString16(*DisplayPtr*, *DrawableID*, *GraphicsContext*, *X*, *Y*, *String*, *Length*)
Display **DisplayPtr*;
Drawable *DrawableID*;
GC *GraphicsContext*;
int *X*, *Y*;
XChar2b **String*;
int *Length*;

FORTRAN Syntax

external fxdrawstring16
integer*4 *DisplayPtr*
integer*4 *DrawableID*, *GraphicsContext*
integer*4 *X*, *Y*, *String*, *Length*
call fxdrawstring16(*DisplayPtr*, *DrawableID*, *GraphicsContext*, *X*, *Y*, *String*, *Length*)

Description

The **XDrawString** subroutine draws 16-bit text characters in a specified destination drawable. Each character image, as defined by the font in the graphics context, is treated as an additional mask for a fill operation on the drawable. The destination drawable is modified only where the font character has a bit set to 1.

The **XDrawString16** subroutine uses the *function, plane_mask, fill_style, font, subwindow_mode, clip_x_origin, clip_y_origin*, and *clip_mask* graphics context fields. It also uses the *foreground, background, tile, stipple, ts_x_origin*, and *ts_y_origin* graphics context mode-dependent fields.

Parameters

DisplayPtr	Specifies the connection to the X Server.
DrawableID	Specifies the drawable.
GraphicsContext	Specifies the graphics context.
Length	Specifies the number of characters in the string parameter.
String	Specifies the character string.
X	Specifies the x coordinate relative to the origin of the specified drawable. This coordinate defines the baseline starting position for the initial character.

Y	Specifies the y coordinate relative to the origin of the specified drawable. This coordinate defines the baseline starting position for the initial character.

Error Codes

BadDrawable

BadFont

BadGC

BadImplementation

BadMatch

Implementation Specifics

This Xlib subroutine is part of AIXwindows Run Time Environment in AIXwindows Environment/6000.

Related Information

The **XChar2b** data structure.

The **PolyText16** protocol.

The **XDrawString** subroutine.

XDrawText Subroutine

Purpose

Draws complex 8-bit characters in a specified drawable.

Libraries

Enhanced X-Windows Library (**libX11.a**)

FORTRAN 77 Library (**libXfx.a**)

C Syntax

XDrawText(*DisplayPtr, DrawableID, GraphicsContext, X, Y, Items, NumberItems***)**
Display **DisplayPtr*;
Drawable *DrawableID*;
GC *GraphicsContext*;
int *X, Y*;
XTextItem **Items*;
int *NumberItems*;

FORTRAN Syntax

external fxdrawtext
integer*4 *DisplayPtr*
integer*4 *DrawableID, GraphicsContext*
integer*4 *X, Y, Items, NumberItems*
call fxdrawtext(*DisplayPtr, DrawableID, GraphicsContext, X, Y, Items, NumberItems***)**

Description

The **XDrawText** subroutine draws 8-bit characters in a specified drawable using the **XTextItem** data structure. It allows complex spacing and font shifts between counted strings.

Each text item is processed in turn. A *Font* variable set to a value other than **None** causes the font to be stored in the graphics context and used in subsequent text.

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 and is not dependent on any characteristics of the font.

Each character image, as defined by the font in the graphics context, is treated as an additional mask for a fill operation on the drawable. The drawable is modified only where the font character has a bit set to 1. If a text item generates a **BadFont** error, the previous text items may have been drawn.

The **XDrawText** subroutine uses the *function, plane_mask, fill_style, font, subwindow_mode, clip_x_origin, clip_y_origin* and *clip_mask* graphics context fields. It also uses the *foreground, background, tile, stipple, ts_x_origin*, and *ts_y_origin* graphics context mode-dependent fields.

Parameters

DisplayPtr	Specifies the connection to the X Server.
DrawableID	Specifies the drawable.

GraphicsContext	Specifies the graphics context.
Items	Specifies a pointer to an array of text items.
NumberItems	Specifies the number of text items in the array.
X	Specifies the x coordinate relative to the origin of the specified drawable. This coordinate defines the origin of the initial character.
Y	Specifies the y coordinate relative to the origin of the specified drawable. This coordinate defines the origin of the initial character.

Error Codes

BadDrawable

BadFont

BadGC

BadImplementation

BadMatch

Implementation Specifics

This Xlib subroutine is part of AIXwindows Run Time Environment in AIXwindows Environment/6000.

Related Information

The **PolyText8** protocol request.

The **XDrawText16** subroutine.

XDrawText16 Subroutine

Purpose

Draws complex 2-byte text in a specified drawable.

Libraries

Enhanced X-Windows Library (**libX11.a**)

FORTRAN 77 Library (**libXfx.a**)

C Syntax

XDrawText16*DisplayPtr, DrawableID,*
GraphicsContext, X,Y,
ItemsNumberItems)
Display **Display*;*
Drawable *Drawable*;
GC *GraphicsContext*;
int *X, Y*;
XTextItem16 **Items*;
int *NumberItems*;

FORTRAN Syntax

external fxdrawtext16
integer*4 *DisplayPtr*
integer*4 *DrawableID, GraphicsContext*
integer*4 *X, Y, Items, NumberItems*
call fxdrawtext(*DisplayPtr, DrawableID,*
GraphicsContext, X, Y,
Items, NumberItems)

Description

The **XDrawText16** subroutine draws 8-bit characters in a specified drawable using the **XTextItem16** data structure. It allows complex spacing and font shifts between counted strings.

Each text item is processed in turn. A *font* field set to a value other than **None** causes the font to be stored in the graphics context and used in subsequent text.

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 and is not dependent on any characteristics of the font.

Each character image, as defined by the font in the graphics context, is treated as an additional mask for a fill operation on the drawable. The drawable is modified only where the font character has a bit set to 1. If a text item generates a **BadFont** error, the previous text items may have been drawn.

For fonts defined with linear indexing rather than 2-byte matrix indexing, each **XChar2b** data structure is interpreted as a 16-bit number with the *Byte1* field as the most-significant byte.

The **XDrawText16** subroutine uses the *function, plane_mask, fill_style, font, subwindow_mode, clip_x_origin, clip_y_origin*, and *clip_mask* graphics context fields. It also uses the *foreground, background, tile, stipple, ts_x_origin*, and *ts_y_origin* graphics context mode-dependent fields.

Parameters

DisplayPtr	Specifies the connection to the X Server.
DrawableID	Specifies the drawable.
GraphicsContext	Specifies the graphics context.
Items	Specifies a pointer to an array of text items.
NumberItems	Specifies the number of text items in the array.
X	Specifies the x coordinates relative to the origin of the specified drawable. This coordinate defines the origin of the initial character.
Y	Specifies the y coordinate relative to the origin of the specified drawable. This coordinate defines the origin of the initial character.

Error Codes

BadDrawable

BadFont

BadGC

BadImplementation

BadMatch

Implementation Specifics

This Xlib subroutine is part of AIXwindows Run Time Environment in AIXwindows Environment/6000.

Related Information

The **XTextItem** data structure.

The **PolyText16** protocol request.

The **XDrawText** subroutine.

XEmptyRegion Subroutine

Purpose

Determines if a specified region is empty.

Libraries

Enhanced X-Windows Library (**libX11.a**)

FORTRAN 77 Library (**libXfx.a**)

C Syntax

Bool XEmptyRegion(*RegionPtr***)**
Region *RegionPtr*;

FORTRAN Syntax

integer*4 fxemptyregion
external fxemptyregion
integer*4 *RegionPtr*
integer*4 *ReasonCode*
ReasonCode = **fxemptyregion(***RegionPtr***)**

Description

The **XEmptyRegion** subroutine determines if a specified region is empty.

Parameter

RegionPtr Specifies the region.

Return Values

False Indicates that the region is not empty.

True Indicates that the region is empty.

Error Code

BadImplementation

Implementation Specifics

This Xlib subroutine is part of AIXwindows Run Time Environment in AIXwindows Environment/6000.

XEnableAccessControl Subroutine

Purpose

Enables access control.

Libraries

Enhanced X-Windows Library (**libX11.a**)

FORTRAN 77 Library (**libXfx.a**)

C Syntax

XEnableAccessControl(*DisplayPtr***)**
Display ***DisplayPtr;

FORTRAN Syntax

external fxenableaccesscontrol
integer*4 *DisplayPtr*
call fxenableaccesscontrol(*DisplayPtr***)**

Description

The **XEnableAccessControl** subroutine 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 or have the necessary permission in the initial authorization at connection setup.

Parameter

DisplayPtr Specifies the connection to the X Server.

Error Codes

BadAccess

BadImplementation

Implementation Specifics

This Xlib subroutine is part of AIXwindows Run Time Environment in AIXwindows Environment/6000.

Related Information

The **SetAccessControl** protocol request.

The **XDisableAccessControl** subroutine.

XEqualRegion Subroutine

Purpose

Determines if two regions are the same.

Libraries

Enhanced X-Windows Library (**libX11.a**)

FORTRAN 77 Library (**libXfx.a**)

C Syntax

Bool XEqualRegion(*Region1*, *Region2***)**
Region *Region1*, *Region2*;

FORTRAN Syntax

integer*4 fxequalregion
external fxequalregion
integer*4 *Region1*, *Region2*
integer*4 *ReasonCode*
ReasonCode = **fxequalregion(***Region1*, *Region2***)**

Description

The **XEqualRegion** subroutine determines if two regions have the same offset, size, and shape.

Parameters

Region1	Specifies one of the two regions to compare for offset, size, and shape.
Region2	Specifies one of the two regions to compare for offset, size, and shape.

Return Values

False	The two regions are not identical.
True	The two regions are identical.

Error Code

BadImplementation

Implementation Specifics

This Xlib subroutine is part of AIXwindows Run Time Environment in AIXwindows Environment/6000.

XEventsQueued Subroutine

Purpose

Checks the number of events in the event queue.

Library

Enhanced X-Windows Library (**libX11.a**)

FORTRAN 77 Library (**libXfx.a**)

C Syntax

int XEventsQueued(*DisplayPtr*, *Mode***)**
**Display ****DisplayPtr*;
int *Mode*;

FORTRAN Syntax

integer*4 fxeventsqueued
external fxeventsqueued
integer*4 *DisplayPtr*
integer*4 *Mode*
integer*4 *ReasonCode*
ReasonCode = **fxeventsqueued(***DisplayPtr*, *Mode***)**

Description

The **XEventsQueued** subroutine checks the number of events in the event queue. It always returns immediately without I/O If there are already events in the queue.

Parameters

DisplayPtr Specifies the connection to the X Server.

Mode Specifies the mode, which can be one of the following values:

QueuedAlready Indicates that the **XEventsQueued** subroutine returns the number of events already in the event queue and does not perform a subroutine. (Specifying this mode is equivalent to using the **XQLength** subroutine.)

QueuedAfterFlush Indicates that the **XEventsQueued** subroutine returns the number of events already in the queue when this number is nonzero. If there are no events in the queue, it: flushes the output buffer; attempts to read more events out of the application's connection; and, returns the number of events read. (Specifying this mode is equivalent to using the **XPending** subroutine.)

QueuedAfterReading Indicates that the **XEventsQueued** subroutine returns the number of events already in the queue when this number is nonzero. If there

are no events in the queue, it: attempts, without flushing the output buffer, to read more events out of the connection to the application; and, returns the number of events read.

Error Code

BadImplementation

Implementation Specifics

This Xlib subroutine is part of AIXwindows Run Time Environment in AIXwindows Environment/6000.

Related Information

The **QLength** subroutine, **XPending** subroutine.

XFetchBuffer Subroutine

Purpose

Gets data from a specified cut buffer.

Libraries

Enhanced X-Windows Library (**libX11.a**)

FORTRAN 77 Library (**libXfx.a**)

C Syntax

char *XFetchBuffer(*DisplayPtr*, *NumberBytesReturn*, *Buffer***)**
**Display ****DisplayPtr*;
**int ****NumberBytesReturn*;
int *Buffer*;

FORTRAN Syntax

integer*4 fxfetchbuffer
external fxfetchbuffer
integer*4 *DisplayPtr*
integer*4 *NumberBytesReturn*, *ReturnBuffer*
integer*4 *FetchBuffer*
FetchBuffer = **fxfetchbuffer(***DisplayPtr*, *NumberBytesReturn*, *ReturnBuffer***)**

Description

The **XFetchBuffer** subroutine returns data from a specified cut buffer. If there is no data in the buffer, it returns **0** in the *NumberBytesReturn* parameter.

Parameters

Buffer	Specifies the buffer from which the stored data will be returned.
DisplayPtr	Specifies the connection to the X Server.
NumberBytesReturn	Returns the number of bytes stored as a string in the buffer.

Return Value

0	There is no data in the cut buffer.

Error Codes

BadImplementation

BadValue

Implementation Specifics

This Xlib subroutine is part of AIXwindows Run Time Environment in AIXwindows Environment/6000.

XFetchBytes Subroutine

Purpose

Gets data from the first cut buffer.

Libraries

Enhanced X-Windows Library (**libX11.a**)

FORTRAN 77 Library (**libXfx.a**)

C Syntax

char *XFetchBytes(*DisplayPtr*, *NumberBytesReturn***)**
**Display ****DisplayPtr*;
**int ****NumberBytesReturn*;

FORTRAN Syntax

integer*4 fxfetchbytes
external fxfetchbytes
integer*4 *DisplayPtr*
integer*4 *NumberBytesReturn*
integer*4 *FetchBytes*
FetchBytes = **fxfetchbytes(***DisplayPtr*, *NumberBytesReturn***)**

Description

The **XFetchBytes** subroutine returns data from cut buffer 0. It returns the number of bytes in the *NumberBytesReturn* parameter. If there is no data in the buffer, it returns a value of **NULL** and sets the *NumberBytesReturn* parameter to 0. The appropriate amount of storage is allocated, and the pointer is returned. The client must free this storage by calling the **XFree** subroutine.

Since the cut buffer does not necessarily contain text, it may contain embedded **NULL** bytes and may not terminate with a **NULL** byte.

Parameters

DisplayPtr	Specifies the connection to the X Server.
NumberBytesReturn	Returns the number of bytes stored as a string in the buffer.

Return Values

NULL	Indicates that there is no data in the buffer.
pointer	Specifies a pointer to the data in the buffer.

Error Code

BadImplementation

Implementation Specifics

This Xlib subroutine is part of AIXwindows Run Time Environment in AIXwindows Environment/6000.

Related Information

The **GetProperty** protocol request.

The **XFree** subroutine.

XFetchName Subroutine

Purpose

Gets the name of a window.

Libraries

Enhanced X-Windows Library (**libX11.a**)

FORTRAN 77 Library (**libXfx.a**)

C Syntax

int XFetchName(*DisplayPtr*, *WindowID*, *WindowNameReturn***)**
Display ******DisplayPtr*;
Window *WindowID*;
char *******WindowNameReturn*;

FORTRAN Syntax

integer*4 fxfetchname
external fxfetchname
integer*4 *DisplayPtr*
integer*4 *WindowID*
integer*4 *WindowNameReturn*
integer*4 *ReturnCode*
ReturnCode = **fxfetchname(***DisplayPtr*, *WindowID*, *WindowNameReturn***)**

Description

The **XFetchName** subroutine gets the name of a specified window.

If the **WM_NAME** property has not been set for this window, the **XFetchName** subroutine sets the *WindowNameReturn* parameter to the value of **NULL**.

After using this subroutine, the client must free the *WindowNameReturn* parameter by using the **XFree** subroutine.

Parameters

DisplayPtr	Specifies the connection to the X Server.
WindowID	Specifies the window ID for the window where the pointer will be returned.
WindowNameReturn	Returns a pointer (a null-terminated string) to the specified window.

Return Values

Non-zero	Indicates that the window name is successfully returned.
0	Indicates that the **XFetchName** subroutine fails (no name was set for the window).

Error Codes

BadWindow

BadImplementation

Implementation Specifics

This Xlib subroutine is part of AIXwindows Run Time Environment in AIXwindows Environment/6000.

Related Information

The **GetProperty** protocol request.

The **XFree** subroutine.

XFillArc Subroutine

Purpose

Fills a single arc in a given drawable.

Libraries

Enhanced X-Windows Library (**libX11.a**)

FORTRAN 77 Library (**libXfx.a**)

C Syntax

XFillArc(*DisplayPtr, DrawableID, GraphicsContext, X, Y, Width, Height, Angle1, Angle2*)
Display***DisplayPtr*;
Drawable *DrawableID*;
GC *GraphicsContext*;
int *X, Y*;
unsigned int *Width, Height*;
int *Angle1, Angle2*;

FORTRAN Syntax

external fxfillarc
integer*4 *DisplayPtr*
integer*4 *DrawableID, GraphicsContext*
integer*4 *X, Y, Width, Height, Angle1, Angle2*
call fxfillarc(*DisplayPtr, DrawableID, GraphicsContext, X, Y, Width, Height, Angle1, Angle2*)

Description

The **XFillArc** subroutine fills a single arc in a given drawable. It fills the region closed by the infinitely thin path described by the specified arc and, depending on the *arc_mode* field specified in the graphics context, one or two line segments. The single line segment joining the endpoints of the arc is used for the **ArcChord** value. The two line segments joining the endpoints of the arc with the center point are used for the **ArcPieSlice** value.

The **XFillArc** subroutine uses the *function, plane_mask, fill_style, arc_mode, subwindow_mode, clip_x_origin, xlip_y_origin,* and *clip_mask* graphics context fields. It also uses the : *foreground, background, tile, stipple, ts_x_origin,* and *ts_y_origin* graphics context mode-dependent fields.

Parameters

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.
DisplayPtr	Specifies the connection to the X Server.
DrawableID	Specifies the drawable.
GraphicsContext	Specifies the graphics context.

Height	Specifies the height, which with the width, defines the major and minor axes of the arc.
Width	Specifies the width, which with the height, defines the major and minor axes of the arc.
X	Specifies the x coordinate, which with the y coordinate, defines the upper-left corner of the rectangle. This coordinate is relative to the origin of the specified drawable.
Y	Specifies the y coordinate, which with the x coordinate, defines the upper-left corner of the rectangle. This coordinate is relative to the origin of the specified drawable.

Error Codes

BadDrawable

BadGC

BadImplementation

BadMatch

Implementation Specifics

This Xlib subroutine is part of AIXwindows Run Time Environment in AIXwindows Environment/6000.

Related Information

The **XArc** data structure.

The **PolyFillArc** protocol request.

The **XFillArcs** subroutine.

XFillArcs Subroutine

Purpose

Fills multiple arcs in a given drawable.

Libraries

Enhanced X-Windows Library (**libX11.a**)

FORTRAN 77 Library (**libXfx.a**)

C Syntax

XFillArcs(*DisplayPtr*, *DrawableID*, *GraphicsContext*, *Arcs*, *NumberArcs*)
Display **DisplayPtr*;
Drawable *DrawableID*;
GC *GraphicsContext*;
XArc **Arcs*;
int *NumberArcs*;

FORTRAN Syntax

external fxfillarcs
integer*4 *DisplayPtr*
integer*4 *DrawableID*, *GraphicsContext*
integer*4 *Arcs*, *NumberArcs*
call fxfillarcs(*DisplayPtr*, *DrawableID*, *GraphicsContext*, *Arcs*, *NumberArcs*)

Description

The **XFillArcs** subroutine fills multiple arcs in the specified drawable in the order listed in the array of the **XArc** data structure. For each arc, it fills the region closed within the path described by the specified arc and either one or two line segments, depending on the *ArcMode* parameter specified in the graphics context. If the *ArcMode* parameter is specified as the **ArcChord** value, the single line segment joining the endpoints of the arc is used. If the *ArcMode* parameter is specified as the **ArcPieSlice** value, the two line segments joining the endpoints of the arc with the center point are used.

For any given arc, no pixel is drawn more than once. If regions intersect, the intersecting pixels are drawn multiple times.

The **XFillArcs** subroutine uses the *function, plane_mask, fill_style, arc_mode, subwindow_mode, clip_x_origin, clip_y_origin*, and *clip_mask* graphics context fields. It also uses the *foreground, background, tile, stipple, ts_x_origin*, and *ts_y_origin* graphics context mode-dependent fields.

Parameters

Arcs	Specifies a pointer to an array of arcs.
DisplayPtr	Specifies the connection to the X Server.
DrawableID	Specifies the drawable.
GraphicsContext	Specifies the graphics context.
NumberArcs	Specifies the number of arcs in the array.

Error Codes

BadDrawable

BadGC

BadImplementation

BadMatch

Implementation Specifics

This Xlib subroutine is part of AIXwindows Run Time Environment in AIXwindows Environment/6000.

Related Information

The **PolyFillArc** protocol request.

The **XFillArc** subroutine.

XFillPolygon Subroutine

Purpose

Fills a polygon-shaped area in a specified drawable.

Libraries

Enhanced X-Windows Library (**libX11.a**)

FORTRAN 77 Library (**libXfx.a**)

C Syntax

XFillPolygon(*DisplayPtr*, *DrawableID*, *GraphicsContext*, *Points*, *NumberPoints*, *Shape*, *Mode*)
Display **DisplayPtr*;
Drawable *DrawableID*;
GC *GraphicsContext*;
XPoint **Points*;
int *NumberPoints*;
int *Shape*;
int *Mode*;

FORTRAN Syntax

external fxfillpolygon
integer*4 *DisplayPtr*
integer*4 *DrawableID*, *GraphicsContext*
integer*4 *Points*, *NumberPoints*, *Shape*, *Mode*
call fxfillpolygon(*DisplayPtr*, *DrawableID*, *GraphicsContext*, *Points*, *NumberPoints*, *Shape*, *Mode*)

Description

The **XFillPolygon** subroutine fills a polygon-shaped area in a specified drawable. It fills a region closed by a 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* parameter may be specified as the **Complex**, **Convex**, or **Nonconvex** value. Specifying the value of **Complex** indicates that the path self-intersects. Specifying the value of **Convex** vindicates the path is totally convex. If the value of **Convex** is specified for a path that is not convex, the graphics results are undefined. Specifying the value of **Nonconvex** indicates the path does not self-intersect, but it is not totally convex. If the value of **Nonconvex** is specified for a self-intersecting path, the graphics results are undefined.

The *FillRule* component of the **GC** controls the filling behavior of self-intersecting polygons.

The *Mode* parameter may be specified as the **CoordModeOrigin** or **CoordModePrevious** value. The first point is always relative to the origin of the drawable. If the value of **CoordModeOrigin** is specified, all points are relative to the origin of the drawable. If **CoordModePrevious** is specified, all points after the first are relative to the previous point.

The **XFillPolygon** subroutine uses the *function, plane_mask, fill_style, fill_rule, subwindow_mode, clip_x_origin, clip_y_origin*, and *clip_mask* graphics context fields. It also uses the *foreground, tile, stipple, ts_x_origin*, and *ts_y_origin* graphics context mode-dependent fields.

Parameters

DisplayPtr	Specifies the connection to the X Server.
DrawableID	Specifies the drawable.
GraphicsContext	Specifies the graphics context.
Mode	Specifies the coordinate mode.
NumberPoints	Specifies the number of points in the array.
Points	Specifies a pointer to an array of points.
Shape	Specifies the shape.

Error Codes

BadDrawable

BadGC

BadImplementation

BadMatch

BadValue

Implementation Specifics

This Xlib subroutine is part of AIXwindows Run Time Environment in AIXwindows Environment/6000.

Related Information

The **FillPoly** protocol request.

XFillRectangle Subroutine

Purpose

Fills a single rectangular-shaped area in specified drawable.

Libraries

Enhanced X-Windows Library (**libX11.a**)

FORTRAN 77 Library (**libXfx.a**)

C Syntax

XFillRectangle(*DisplayPtr*, *DrawableID*, *GraphicsContext*, *X*, *Y*, *Width*, *Height*)
Display **DisplayPtr*;
Drawable *DrawableID*;
GC *GraphicsContext*;
int *X*, *Y*;
unsigned int *Width*, *Height*;

FORTRAN Syntax

external fxfillrectangle
integer*4 *DisplayPtr*
integer*4 *DrawableID*, *GraphicsContext*
integer*4 *X*, *Y*, *Width*, *Height*
call fxfillrectangle(*DisplayPtr*, *DrawableID*, *GraphicsContext*, *X*, *Y*, *Width*, *Height*)

Description

The **XFillRectangle** subroutine fills a single rectangular-shaped area as if a four-point **FillPolygon** protocol request was specified. For example:

```
[X,Y] [X+Width,Y] [X+Width,Y+Height] [X,Y+Height]
```

The **XFillRectangle** subroutine does not draw a pixel more than once. If rectangles intersect, the intersecting pixels are drawn multiple times.

This subroutine uses the *function, plane_mask, fill_style, fill_rule, subwindow_mode, clip_x_origin, clip_y_origin*, and *clip_mask* graphics context fields. It also uses the *foreground, background, tile, stipple, ts_x_origin*, and *ts_y_origin* graphics context mode-dependent fields.

Parameters

DisplayPtr	Specifies the connection to the X Server.
DrawableID	Specifies the drawable.
GraphicsContext	Specifies the graphics context.
Height	Specifies the height, which defines the dimensions of the rectangle.
Width	Specifies the width, which defines the dimensions of the rectangle.
X	Specifies the x coordinate relative to the origin of the specified drawable, and defining the upper-left corner of the rectangle.

Y	Specifies the y coordinate relative to the origin of the specified drawable, and defining the upper-left corner of the rectangle.

Error Codes

BadDrawable

BadGC

BadImplementation

BadMatch

Implementation Specifics

This Xlib subroutine is part of AIXwindows Run Time Environment in AIXwindows Environment/6000.

Related Information

The **PolyFillRectangle** protocol request.

The **XFillRectangles** subroutine.

XFillRectangles Subroutine

Purpose

Fills multiple rectangular-shaped areas in a specified drawable.

Libraries

Enhanced X-Windows Library (**libX11.a**)

FORTRAN 77 Library (**libXfx.a**)

C Syntax

XFillRectangles(*DisplayPtr*, *DrawableID*, *GraphicsContext*, *Rectangles*, *NumberRectangles*)
Display **DisplayPtr*;
Drawable *DrawableID*;
GC *GraphicsContext*;
XRectangle **Rectangles*;
int *NumberRectangles*;

FORTRAN Syntax

external fxfillrectangles
integer*4 *DisplayPtr*
integer*4 *DrawableID*, *GraphicsContext*
integer*4 *Rectangles*, *NumberRectangles*
call fxfillrectangles(*DisplayPtr*, *DrawableID*, *GraphicsContext*, *Rectangles*, *NumberRectangles*)

Description

The **XFillRectangles** subroutine fills multiple rectangular-shaped areas in a specified drawable as if a four-point **FillPolygon** protocol request was specified for each rectangle. For example:

```
[X,Y] [X+Width,Y] [X+Width,Y+Height] [X,Y+Height]
```

The **XFillRectangles** subroutine fills rectangles in the order listed in the array of an **XRectangle** data structure.

For any given rectangle, no pixel is drawn more than once. If rectangles intersect, the intersecting pixels are drawn multiple times.

The **XFillRectangles** subroutine uses the *function*, *plane_mask*, *fill_style*, *fill_rule*, *subwindow_mode*, *clip_x_origin*, *clip_y_origin*, and *clip_mask* graphics context fields. It also uses the *foreground*, *background*, *tile*, *stipple*, *ts_x_origin*, and *ts_y_origin* graphics context mode-dependent fields.

Parameters

DisplayPtr	Specifies the connection to the X Server.
DrawableID	Specifies the drawable.
GraphicsContext	Specifies the graphics context.
NumberRectangles	Specifies the number of rectangles in the array.
Rectangles	Specifies a pointer to an array of rectangles.

Error Codes

BadDrawable

BadGC

BadImplementation

BadMatch

Implementation Specifics

This Xlib subroutine is part of AIXwindows Run Time Environment in AIXwindows Environment/6000.

Related Information

The **PolyFillRectangle** protocol request.

The **XFillRectangle** subroutine.

XFindContext Subroutine

Purpose

Gets the data, including its context type, associated with window.

Libraries

Enhanced X-Windows Library (**libX11.a**)

FORTRAN 77 Library (**libXfx.a**)

C Syntax

int XFindContext(*DisplayPtr*, *WindowID*, *Context*, *DataReturn***)**
Display **DisplayPtr*;
Window *WindowID*;
XContext *Context*;
caddr_t **DataReturn*;

FORTRAN Syntax

integer*4 fxfindcontext
external fxfindcontext
integer*4 *DisplayPtr*, *WindowID*, *Context*
integer*4 *DataReturn*
integer*4 *Status*
Status = **fxfindcontext(***DisplayPtr*, *WindowID*, *Context*, *DataReturn***)**

Description

The **XFindContext** subroutine gets the data, including its context type, associated with a window.

Parameters

Context	Specifies the context type to which the data belongs.
DataReturn	Returns a pointer to the data.
DisplayPtr	Specifies the connection to the X Server.
WindowID	Specifies the window ID for the window with which the data is associated.

Return Values

0	The **XFindContext** subroutine executes successfully.
Nonzero	The subroutine cannot execute.

Error Codes

BadImplementation

XCNOENT (context-not-found)

Implementation Specifics

This Xlib subroutine is part of AIXwindows Run Time Environment in AIXwindows Environment/6000.

XFlush Subroutine

Purpose

Flushes the output buffer.

Libraries

Enhanced X-Windows Library (**libX11.a**)

FORTRAN 77 Library (**libXfx.a**)

C Syntax

XFlush(*DisplayPtr***)**
Display ***DisplayPtr;*

FORTRAN Syntax

external fxflush
integer*4 *DisplayPtr*
call fxflush(*DisplayPtr***)**

Description

The **XFlush** subroutine flushes the output buffer. Most client applications do not require this subroutine because the output buffer is automatically flushed as needed by calls to the **XPending**, **XNextEvent**, or **XWindowEvent** subroutine. Events generated by the server may be placed in the library's event queue.

Parameter

DisplayPtr Specifies the connection to the X Server.

Error Code

BadImplementation

Implementation Specifics

This Xlib subroutine is part of AIXwindows Run Time Environment in AIXwindows Environment/6000.

Related Information

The **XNextEvent** subroutine, **XPending** subroutine, **XWindowEvent** subroutine.

XForceScreenSaver Subroutine

Purpose

Forces the screen saver on or off.

Libraries

Enhanced X-Windows Library (**libX11.a**)

FORTRAN 77 Library (**libXfx.a**)

C Syntax

XForceScreenSaver(*DisplayPtr*, *Mode***)**
Display **DisplayPtr*;
int *Mode*;

FORTRAN Syntax

external fxforcescreensaver
integer*4 *DisplayPtr*
integer*4 *Mode*
call fxforcescreensaver(*DisplayPtr*, *Mode***)**

Description

The **XForceScreenSaver** subroutine forces the screen saver on or off by applying a mode to the screen saver.

The *Mode* parameter may be specified as the **ScreenSaverActive** or **ScreenSaverReset** value. Specifying the value of **ScreenSaverActive** activates the screen saver even if the screen saver is currently disabled. If the screen saver is currently enabled, specifying the value of **ScreenSaverReset** deactivates the screen saver, as if device input has been received. It also resets the activation timer.

Parameters

DisplayPtr	Specifies the connection to the X Server.
Mode	Specifies the screen saver mode.

Error Codes

BadValue

BadImplementation

Implementation Specifics

This Xlib subroutine is part of AIXwindows Run Time Environment in AIXwindows Environment/6000.

Related Information

The **ForceScreenSaver** protocol request.

The **XSetScreenSaver** subroutine.

XFree Subroutine

Purpose

Frees in-memory data created by an **Xlib** subroutine.

Libraries

Enhanced X-Windows Library (**libX11.a**)

FORTRAN 77 Library (**libXfx.a**)

C Syntax

XFree(*Data***)**
char ****Data*;**

FORTRAN Syntax

external fxfree
integer*4 *Data*
call fxfree(*Data***)**

Description

The **XFree** subroutine frees the specified in-memory data allocated by any **Xlib** subroutine.

Parameter

Data	Specifies a pointer to the data that is to be freed.

Error Code

BadImplementation

Implementation Specifics

This Xlib subroutine is part of AIXwindows Run Time Environment in AIXwindows Environment/6000.

XFreeColormap Subroutine

Purpose

Deletes the association between the specified colormap ID and the colormap.

Libraries

Enhanced X-Windows Library (**libX11.a**)

FORTRAN 77 Library (**libXfx.a**)

C Syntax

XFreeColormap(*DisplayPtr*, *ColormapID*)
Display **DisplayPtr*;
Colormap *ColormapID*;

FORTRAN Syntax

external fxfreecolormap
integer*4 *DisplayPtr*
integer*4 *ColormapID*
call fxfreecolormap(*DisplayPtr*, *ColormapID*)

Description

The **XFreeColormap** subroutine deletes the association between a colormap resource ID and colormap. It also frees the storage for this colormap but has no effect on the default colormap for the screen.

If the *Colormap* parameter specifies an installed colormap for a screen, the **XFreeColormap** subroutine uninstalls this colormap.

If the *Colormap* parameter specifies the colormap for a window, the **XFreeColormap** subroutine changes the colormap associated with the window to the value of **None** and generates a **ColormapNotify** event. The colors displayed for a window with a colormap of the value of **None** are not defined.

Parameters

ColormapID Specifies the colormap to be deleted.

DisplayPtr Specifies the connection to the X Server.

Error Codes

BadColor

BadImplementation

Implementation Specifics

This Xlib subroutine is part of AIXwindows Run Time Environment in AIXwindows Environment/6000.

Related Information

The **FreeColormap** protocol request.

The **XUninstallColormap** subroutine.

XFreeColors Subroutine

Purpose

Frees colormap cells.

Libraries

Enhanced X-Windows Library (**libX11.a**)

FORTRAN 77 Library (**libXfx.a**)

C Syntax

XFreeColors(*DisplayPtr*, *ColormapID*, *Pixels*, *NumberPixels*, *Planes*)
Display **DisplayPtr*;
Colormap *ColormapID*;
unsigned long *Pixels*[];
int *NumberPixels*;
unsigned long *Planes*;

FORTRAN Syntax

external fxfreecolors
integer*4 *DisplayPtr*
integer*4 *ColormapID*
integer*4 *Pixels*, *NumberPixels*, *Planes*
call fxfreecolors(*DisplayPtr*, *ColormapID*, *Pixels*, *NumberPixels*, *Planes*)

Description

The **XFreeColors** subroutine frees colormap cells represented by pixels whose values are in the pixels array. The **XFreeColors** subroutine frees the pixels that were allocated by the client using the **XAllocColor**, **XAllocColorCells**, **XAllocColorPlanes**, and **XAllocNamedColor** subroutines.

Freeing a pixel obtained from the **XAllocColorPlanes** subroutine may not allow the pixel to be reused until all the related pixels are also freed.

The *Planes* parameter should not have bits in common with the pixels. The set of all pixels is produced by combining the subsets of the *Planes* parameter with the pixels.

All specified pixels that are allocated by the colormap are freed, even if one or more pixels produce an error. If more than one pixel is in error, the one reported is arbitrary. If a specified pixel is not in the colormap, a **BadValue** error results. If a specified pixel is unallocated, or is only allocated by another client, a **BadAccess** error results.

Parameters

ColormapID	Specifies the colormap ID.
DisplayPtr	Specifies the connection to the X Server.
NumberPixels	Specifies the number of pixels.
Pixels	Specifies an array of pixel values that map to the cells in the specified colormap.
Planes	Specifies the planes to be freed.

Error Codes

BadAccess

BadColor

BadImplementation

BadValue

Implementation Specifics

This Xlib subroutine is part of AIXwindows Run Time Environment in AIXwindows Environment/6000.

Related Information

The **FreeColors** protocol request.

The **XAllocColor** subroutine, **XAllocColorCells** subroutine, **XAllocColorPlanes** subroutine, **XAllocNamedColor** subroutine.

XFreeCursor Subroutine

Purpose

Deletes the association between the specified cursor ID and the cursor.

Libraries

Enhanced X-Windows Library (**libX11.a**)

FORTRAN 77 Library (**libXfx.a**)

C Syntax

XFreeCursor(*DisplayPtr*, *CursorID*)
Display **DisplayPtr*;
Cursor *CursorID*;

FORTRAN Syntax

external fxfreecursor
integer*4 *DisplayPtr*
integer*4 *CursorID*
call fxfreecursor(*DisplayPtr*, *CursorID*)

Description

The **XFreeCursor** subroutine 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.

Parameters

CursorID Specifies the cursor.

DisplayPtr Specifies the connection to the X Server.

Error Codes

BadCursor

BadImplementation

Implementation Specifics

This Xlib subroutine is part of AIXwindows Run Time Environment in AIXwindows Environment/6000.

Related Information

The **FreeCursor** protocol request.

XFreeFont Subroutine

Purpose

Deletes the association between the font ID and the font.

Libraries

Enhanced X-Windows Library (**libX11.a**)

FORTRAN 77 Library (**libXfx.a**)

C Syntax

XFreeFont(*DisplayPtr*, *FontStructure*)
Display ***DisplayPtr**;
XFontStruct ***FontStructure**;

FORTRAN Syntax

external fxfreefont
integer*4 *DisplayPtr*
integer*4 *FontStructure*
call fxfreefont(*DisplayPtr*, *FontStructure*)

Description

The **XFreeFont** subroutine deletes the association between the font resource ID and the specified font. The specified font is freed when no other resources reference it. The data and the font should not be referenced again.

Parameters

DisplayPtr	Specifies the connection to the X Server.
FontStructure	Specifies the storage associated with the font.

Error Codes

BadFont

BadImplementation

Implementation Specifics

This Xlib subroutine is part of AIXwindows Run Time Environment in AIXwindows Environment/6000.

Related Information

The **XChar2b** data structure.

The **CloseFont** protocol request.

XFreeFontInfo Subroutine

Purpose

Frees the font information array.

Libraries

Enhanced X-Windows Library (**libX11.a**)

FORTRAN 77 Library (**libXfx.a**)

C Syntax

XFreeFontInfo(*Names*, *FontInfo*, *ActualCount*)
char ***Names*;
XFontStruct **FontInfo*;
int *ActualCount*;

FORTRAN Syntax

external fxfreefontinfo
integer*4 *Names*
integer*4 *FontInfo*, *ActualCount*
call fxfreefontinfo(*Names*, *FontInfo*, *ActualCount*)

Description

The **XFreeFontInfo** subroutine frees the font information array returned by the **XListFontsWithInfo** subroutine.

Parameters

ActualCount	Specifies the actual number of matched font names.
FontInfo	Specifies a pointer to the specified font information.
Names	Specifies a pointer to the specified list of font names.

Error Code

BadImplementation

Implementation Specifics

This Xlib subroutine is part of AIXwindows Run Time Environment in AIXwindows Environment/6000.

Related Information

The **XChar2b** data structure, **XFontStruct** data structure.

The **XListFontsWithInfo** subroutine.

XFreeFontNames Subroutine

Purpose

Frees a font name list.

Libraries

Enhanced X-Windows Library (**libX11.a**)

FORTRAN 77 Library (**libXfx.a**)

C Syntax

XFreeFontNames(*List***)**
char ****List*[];**

FORTRAN Syntax

external fxfreefontnames
integer*4 *List*
call fxfreefontnames(*List***)**

Description

The **XFreeFontNames** subroutine frees the array and strings returned by the **XListFonts** or **XListFontsWithInfo** subroutine.

Parameter

List Specifies the array of strings to be freed.

Error Code

BadImplementation

Implementation Specifics

This Xlib subroutine is part of AIXwindows Run Time Environment in AIXwindows Environment/6000.

Related Information

The **XChar2b** data structure.

The **XListFonts** subroutine, **XListFontsWithInfo** subroutine.

XFreeFontPath Subroutine

Purpose

Frees data allocated by the **XGetFontPath** subroutine.

Libraries

Enhanced X-Windows Library (**libX11.a**)

FORTRAN 77 Library (**libXfx.a**)

C Syntax

XFreeFontPath(*List*)
char ***List*;

FORTRAN Syntax

external fxfreefontpath
integer*4 *List*
call fxfreefontpath(*List*)

Description

The **XFreeFontPath** subroutine frees data allocated by the **XGetFontPath** subroutine.

Parameter

List Specifies the array of strings to be freed.

Error Code

BadImplementation

Implementation Specifics

This Xlib subroutine is part of AIXwindows Run Time Environment in AIXwindows Environment/6000.

Related Information

The **XChar2b** data structure.

The **XGetFontPath** subroutine.

XFreeGC Subroutine

Purpose

Deletes the association between the specified graphics context ID and the graphics context.

Libraries

Enhanced X-Windows Library (**libX11.a**)

FORTRAN 77 Library (**libXfx.a**)

C Syntax

XFreeGC(*DisplayPtr*, *GraphicsContextID*)
**Display ****DisplayPtr*;
GC *GraphicsContextID*;

FORTRAN Syntax

external fxfreegc
integer*4 *DisplayPtr*
integer*4 *GraphicsContextID*
call fxfreegc(*DisplayPtr*, *GraphicsContextID*)

Description

The **XFreeGC** subroutine deletes the specified graphics context and frees all associated storage.

Parameters

DisplayPtr	Specifies the connection to the X Server.
GraphicsContextID	Specifies the graphics context.

Error Codes

BadGC

BadImplementation

Implementation Specifics

This Xlib subroutine is part of AIXwindows Run Time Environment in AIXwindows Environment/6000.

Related Information

The **FreeGC** protocol request.

XFreeModifiermap Subroutine

Purpose

Deletes an **XModifierKeymap** data structure.

Libraries

Enhanced X-Windows Library (**libX11.a**)

FORTRAN 77 Library (**libXfx.a**)

C Syntax

XFreeModifiermap(*Modifiermap***)**
**XModifierKeymap ****Modifiermap*;

FORTRAN Syntax

external fxfreemodifiermap
integer*4 *ModifierMap*
call fxfreemodifiermap(*ModifierMap***)**

Description

The **XFreeModifiermap** subroutine frees the specified **XModifierKeymap** data structure.

Parameter

Modifiermap Specifies a pointer to the **XModifierKeymap** data structure.

Error Code

BadImplementation

Implementation Specifics

This Xlib subroutine is part of AIXwindows Run Time Environment in AIXwindows Environment/6000.

Related Information

The **XModifierKeymap** data structure.

XFreePixmap Subroutine

Purpose

Deletes the association between the specified pixmap ID and the pixmap.

Libraries

Enhanced X-Windows Library (**libX11.a**)

FORTRAN 77 Library (**libXfx.a**)

C Syntax

XFreePixmap(*DisplayPtr*, *PixmapID*)
Display **DisplayPtr*;
Pixmap *PixmapID*;

FORTRAN Syntax

external fxfreepixmap
integer*4 *DisplayPtr*
integer*4 *PixmapID*
call fxfreepixmap(*DisplayPtr*, *PixmapID*)

Description

The **XFreePixmap** subroutine deletes the association between a pixmap ID and a pixmap. When there are no other associations to the pixmap, it frees all storage associated with the specified pixmap. The pixmap should not be referenced again.

Parameters

DisplayPtr Specifies the connection to the X Server.

PixmapID Specifies the pixmap.

Error Codes

BadPixmap

BadImplementation

Implementation Specifics

This Xlib subroutine is part of AIXwindows Run Time Environment in AIXwindows Environment/6000.

Related Information

The **FreePixmap** protocol request.

XGContextFromGC Subroutine

Purpose

Gets the **GContext** resource ID for a graphics context.

Libraries

Enhanced X-Windows Library (**libX11.a**)

FORTRAN 77 Library (**libXfx.a**)

C Syntax

GContext XGContextFromGC(*GraphicsContextIDt***)**
GC *GraphicsContextID*;

FORTRAN Syntax

integer*4 fxgcontextfromgc
external fxgcontextfromgc
integer*4 *GraphicsContextID*
integer*4 *GraphicsContextID*
GraphicsContextID = **fxgcontextfromgc(***GraphicsContextID***)**

Description

The **XGContextFromGC** subroutine obtains the **GContext** resource ID for the specified graphics context.

Parameter

GraphicsContext Specifies the graphics context.

Error Code

BadImplementation

Implementation Specifics

This Xlib subroutine is part of AIXwindows Run Time Environment in AIXwindows Environment/6000.

Related Information

The **XChar2b** data structure.

XGeometry Subroutine

Purpose

Parses window geometry, given a user-specified position and a default position.

Libraries

Enhanced X-Windows Library (**libX11.a**)

FORTRAN 77 Library (**libXfx.a**)

C Syntax

int XGeometry(*DisplayPtr, ScreenNumber, Position, DefaultPosition, BorderWidth, FontWidth, FontHeight, XAdder, YAdder, XReturn, YReturn, WidthReturn, HeightReturn*

Display **DisplayPtr*;
int *ScreenNumber*;
char **Position*, **DefaultPosition*;
unsigned int *Borderwidth*;
unsigned int *FontWidth, FontHeight*;
int *XAdder, YAdder*;
int **XReturn*, **YReturn*;
int **WidthReturn*, **HeightReturn*;

FORTRAN Syntax

integer*4 fxgeometry
external fxgeometry
integer*4 *DisplayPtr*
integer*4 *ScreenNumber*
character*256 *Position, DefaultPosition*
integer*4 *BorderWidth, FontWidth, FontHeight*
integer*4 *XAdder, YAdder*
integer*4 *XReturn, YReturn*
integer*4 *WidthReturn, HeightReturn*
integer*4 *ChangeMask*
ChangeMask = **fxgeometry(***DisplayPtr, ScreenNumber, Position, DefaultPosition, BorderWidth, FontWidth, FontHeight, XAdder, YAdder, XReturn, YReturn, WidthReturn, HeightReturn*)

Description

The **XGeometry** subroutine determines the placement of a window using the current format as specified by the **XParseGeometry** subroutine, in addition to any user-specified information.

Provided the default geometry specification is fully qualified, if the user-specified geometry is incomplete, the **XGeometry** subroutine will return a bitmask value as defined in the **XParseGeometry** subroutine.

The width and height specified by the *DefaultPosition* parameter will be overridden by user-specified input to the *Position* parameter. The width and height are not affected by the *FontWidth*, *FontHeight*, *Xadder*, or *Yadder* parameter values. The X and Y coordinates equal the width and height from the geometry specifications multiplied by the border width, the screen width and height, padding as specified by the *Xadder* and *Yadder* parameter values, and the *FontHeight* and *FontWidth* parameter values.

Parameters

BorderWidth	Specifies the border width.
DefaultPosition	Specifies the geometry specifications.
DisplayPtr	Specifies the connection to the X Server.
FontHeight	Specifies the font height in pixels (increment size).
FontWidth	Specifies the font width in pixels (increment size).
HeightReturn	Returns the height determined.
Position	Specifies the geometry specifications.
ScreenNumber	Specifies the screen number of the display.
WidthReturn	Returns the width determined.
XAdder	Specifies additional interior padding needed in the window.
YAdder	Specifies additional interior padding needed in the window
XReturn	Returns the X offset.
YReturn	Returns the Y offset.

Error Code

BadImplementation

Implementation Specifics

This Xlib subroutine is part of AIXwindows Run Time Environment in AIXwindows Environment/6000.

Related Information

The **XParseGeometry** subroutine.

XGetAtomName Subroutine

Purpose

Gets the name of a specified atom identifier.

Libraries

Enhanced X-Windows Library (**libX11.a**)

FORTRAN 77 Library (**libXfx.a**)

C Syntax

char *XGetAtomName(*DisplayPtr*, *AtomID*)
**Display ***DisplayPtr*;
Atom *AtomID*;

FORTRAN Syntax

character*256 fxgetatomname
external fxgetatomname
integer*4 *DisplayPtr*
integer*4 *AtomID*
character*256 *Name*
Name = **fxgetatomname(***DisplayPtr*, *AtomID*)

Description

The **XGetAtomName** subroutine returns the name for a specified atom identifier. To free the resulting string, use the **XFree** subroutine.

Parameters

AtomID	Specifies the atom for the property name to be returned.
DisplayPtr	Specifies the connection to the X Server.

Error Codes

BadAtom

BadImplementation

Implementation Specifics

This Xlib subroutine is part of AIXwindows Run Time Environment in AIXwindows Environment/6000.

Related Information

The **GetAtomName** protocol request.

The **XFree** subroutine.

XGetClassHint Subroutine

Purpose

Gets the class of a window.

Libraries

Enhanced X-Windows Library (**libX11.a**)

FORTRAN 77 Library (**libXfx.a**)

C Syntax

Status XGetClassHint(*DisplayPtr*, *WindowID*, *ClassHintsReturn***)**
Display ******DisplayPtr*;
Window *WindowID*;
XClassHint ******ClassHintsReturn*;

FORTRAN Syntax

integer*4 fxgetclasshint
external fxgetclasshint
integer*4 *DisplayPtr*
integer*4 *WindowID*, *ClassHintsReturn*
integer*4 *ReasonCode*
ReasonCode **= fxgetclasshint(***DisplayPtr*, *WindowID*, *ClassHintsReturn***)**

Description

The **XGetClassHint** subroutine gets the class of the specified window.

The **XFree** subroutine frees the *res_name* field and the *res_class* field of the **XClassHints** structure.

Parameters

DisplayPtr	Specifies the connection to the X Server.
WindowID	Specifies the window.
ClassHintsReturn	Returns the **XClassHint** data structure.

Error Codes

BadImplementation

BadWindow

Implementation Specifics

This Xlib subroutine is part of AIXwindows Run Time Environment in AIXwindows Environment/6000.

Related Information

The **XClassHint** data structure.

The **XFree** subroutine.

XGetDefault Subroutine

Purpose

Gets the window option defaults.

Libraries

Enhanced X-Windows Library (**libX11.a**)

FORTRAN 77 Library (**libXfx.a**)

C Syntax

char *XGetDefault(*DisplayPtr*, *Program*, *Option***)**
**Display ****DisplayPtr*;
**char ****Program*;
**char ****Option*;

FORTRAN Syntax

character*256 fxgetdefault
external fxgetdefault
integer*4 *DisplayPtr*
character*256 *Program*
character*256 *Option*
character*256 *GetDefault*
GetDefault = **fxgetdefault(***DisplayPtr*, *Program*, *Option***)**

Description

The **XGetDefault** subroutine helps the client determine the fonts, colors, and other environment defaults favored by a particular user. The strings returned by the **XGetDefault** subroutine are owned by the **Xlib** library 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 is the **$HOME/.Xdefaults** file.

After loading these defaults, the **XGetDefault** subroutine merges additional defaults specified by the **XENVIRONMENT** environment variable. If the **XENVIRONMENT** defaults are defined, they contains a full path name for the additional resource file. If the **XENVIRONMENT** valuable defaults are not defined, the **XGetDefault** subroutine looks for the **$HOME/.Xdefaults-***Name* resource file in the user's home directory. The *Name* parameter specifies the name of the system running the application.

Parameters

DisplayPtr	Specifies the connection to the X Server.
Option	Specifies the option name.
Program	Specifies the program name for the **Xlib** library defaults (usually **argv[0]** of the main program).

Return Values

NULL	Indicates that the option name specified does not exist.
String	Indicates that the option value.

Error Code

BadImplementation

Implementation Specifics

This Xlib subroutine is part of AIXwindows Run Time Environment in AIXwindows Environment/6000.

XGetErrorDatabaseText Subroutine

Purpose

Gets error messages from the error database.

Libraries

Enhanced X-Windows Library (**libX11.a**)

FORTRAN 77 Library (**libXfx.a**)

C Syntax

XGetErrorDatabaseText(*DisplayPtr*, *Name*, *Message*, *DefaultString*, *BufferReturn*, *Length*)
Display *DisplayPtr*;
char **Name*, **Message*;
char **DefaultString*;
char **BufferReturn*;
int *Length*;

FORTRAN Syntax

external fxgeterrordatabasetext
integer*4 *DisplayPtr*
character*256 *Name*
character*256 *Message*
character*256 *DefaultString*
character*256 *BufferReturn*
integer*4 *Length*
call fxgeterrordatabasetext(*DisplayPtr*, *Name*, *Message*, *DefaultString*, *BufferReturn*, *Length*)

Description

The **XGetErrorDatabaseText** subroutine returns a message or the default message from the error message database. The error message database file is the **/usr/lpp/X11/messages/XErrorDB** file on an AIX-based system.

The *Name* parameter generally specifies the name of the application. The *Message* parameter indicates which type of error message to use. The **Xlib** library uses three predefined, case-sensitive message types:

XProtoError	The protocol error number is used as a search string for the *Message* parameter.
XlibMessage	The message strings used internally by the **Xlib** library.
XRequest	The major request protocol number is used for the *Message* parameter.

If no string is found in the error database, the *DefaultString* parameter is returned to the *BufferReturn* parameter.

Parameters

BufferReturn	Returns the error description.
DefaultString	Specifies the default error message if none is found in the database.
DisplayPtr	Specifies the connection to the X Server.
Length	Specifies the size of the buffer.
Message	Specifies the type of error message.
Name	Specifies the application name.

Error Code

BadImplementation

Implementation Specifics

This Xlib subroutine is part of AIXwindows Run Time Environment in AIXwindows Environment/6000.

XGetErrorText Subroutine

Purpose

Gets the error text for a specified error code.

Libraries

Enhanced X-Windows Library (**libX11.a**)

FORTRAN 77 Library (**libXfx.a**)

C Syntax

XGetErrorText(*DisplayPtr*, *Code*, *BufferReturn*, *Length*)
Display **DisplayPtr*;
int *Code*;
char **BufferReturn*;
int *Length*;

FORTRAN Syntax

external fxgeterrortext
integer*4 *DisplayPtr*
integer*4 *Code*
character*256 *BufferReturn*
integer*4 *Length*
call fxgeterrortext(*DisplayPtr*, *Code*, *BufferReturn*, *Length*)

Description

The **XGetErrorText** subroutine copies a null-terminated string describing the specified error code into the specified buffer. Use of this subroutine is recommended, as extensions to the **Xlib** library may define their own error codes and error strings.

Parameters

BufferReturn	Returns the error text.
Code	Specifies the error code for which to obtain error text.
DisplayPtr	Specifies the connection to the X Server.
Length	Specifies the size of the buffer.

Error Code

BadImplementation

Implementation Specifics

This Xlib subroutine is part of AIXwindows Run Time Environment in AIXwindows Environment/6000.

XGetFontPath Subroutine

Purpose

Gets the current font search path.

Libraries

Enhanced X-Windows Library (**libX11.a**)

FORTRAN 77 Library (**libXfx.a**)

C Syntax

char **XGetFontPath(*DisplayPtr*, *NumberPathsReturn***)**
**Display ****DisplayPtr*;
**int ****NumberPathsReturn*;

FORTRAN Syntax

integer*4 fxgetfontpath
external fxgetfontpath
integer*4 *DisplayPtr*
integer*4 *NumberPathsReturn*
integer*4 *Path*
Path = **fxgetfontpath(***DisplayPtr*, *NumberPathsReturn***)**

Description

The **XGetFontPath** subroutine allocates and returns an array of strings containing the search path. The data in the font path should be freed by using the **XFreeFontPath** subroutine when it is no longer needed.

Parameters

DisplayPtr	Specifies the connection to the X Server.
NumberPathsReturn	Returns the number of strings in the font path array.

Error Code

BadImplementation

Implementation Specifics

This Xlib subroutine is part of AIXwindows Run Time Environment in AIXwindows Environment/6000.

Related Information

The **XChar2b** data structure.

The **XFreeFontPath** subroutine.

The **GetFontPath** protocol request.

XGetFontProperty Subroutine

Purpose

Gets a specified font property.

Libraries

Enhanced X-Windows Library (**libX11.a**)

FORTRAN 77 Library (**libXfx.a**)

C Syntax

Bool XGetFontProperty(*FontStructure*, *AtomID*, *ValueReturn***)**
XFontStruct **FontStructure*;
Atom *AtomID*;
unsigned long **ValueReturn*;

FORTRAN Syntax

integer*4 fxgetfontproperty
external fxgetfontproperty
integer*4 *FontStructure*
integer*4 *AtomID*, *ValueReturn*
integer*4 *ReturnCode*
ReturnCode = **fxgetfontproperty(***FontStructure*, *AtomID*, *ValueReturn***)**

Description

The **XGetFontProperty** subroutine returns the value of a specified font property. There is a set of predefined atoms for font properties in the **<X11/Xatom.h>** file. This set contains the standard properties associated with a font.

Parameters

AtomID	Specifies the atom for the property name to be returned.
FontStructure	Specifies the storage associated with the font.
ValueReturn	Returns the value of the font property.

Return Values

True	Indicates that the font property is found.
False	Indicates that the font property cannot be found.

Error Code

BadImplementation

Implementation Specifics

This Xlib subroutine is part of AIXwindows Run Time Environment in AIXwindows Environment/6000.

Related Information

The **XChar2b** data structure, **XFontStruct** data structure.

XGetGeometry Subroutine

Purpose

Gets the current geometry of a specified drawable.

Libraries

Enhanced X-Windows Library (**libX11.a**)

FORTRAN 77 Library (**libXfx.a**)

C Syntax

Status XGetGeometry(*DisplayPtr*, *DrawableID*, *RootReturn*, *XReturn*,*YReturn*, *WidthReturn*, *HeightReturn*,*BorderWidthReturn*, *DepthReturn*)
Display **DisplayPtr*;
Drawable *DrawableID*;
Window **RootReturn*;
int **XReturn*, **YReturn*;
unsigned int **WidthReturn*, **HeightReturn*;
unsigned int **BorderWidthReturn*;
unsigned int **DepthReturn*;

FORTRAN Syntax

integer*4 fxgetgeometry
external fxgetgeometry
integer*4 *DisplayPtr*
integer*4 *DrawableID*
integer*4 *RootReturn*
integer*4 *XReturn*, *YReturn*
integer*4 *WidthReturn*, *HeightReturn*
integer*4 *BorderWidthReturn*
integer*4 *DepthReturn*
integer*4 *Status*
Status = **fxgetgeometry**(*DisplayPtr*, *DrawableID*, *RootReturn*, *XReturn*, *YReturn*, *WidthReturn*, *HeightReturn*, *BorderWidthReturn*, *DepthReturn*)

Description

The **XGetGeometry** subroutine gets the root ID and current geometry of the specified drawable.

The **XGetGeometry** subroutine can be used with a window that has an of **InputOnly** class.

Parameters

BorderWidthReturn	Returns the border width in pixels. If the drawable is a pixmap, it returns 0.
DepthReturn	Returns the depth of the drawable in bits per pixel.
DisplayPtr	Specifies the connection to the X Server.
DrawableID	Specifies the drawable, which may be either a window or a pixmap.

HeightReturn	Returns the height of the inside of the drawable, excluding the border.
RootReturn	Returns the root window ID.
WidthReturn	Returns the width of the inside of the drawable, excluding the border.
XReturn	Returns the x coordinate defining the location of the drawable. If the drawable is a window, this coordinate specifies the upper-left outer corner relative to the origin of the parent. If it is a pixmap, this coordinate is always 0.
YReturn	Returns the y coordinate defining the location of the drawable. If the drawable is a window, this coordinate specifies the upper-left outer corner relative to the origin of the parent. If it is a pixmap, this coordinate is always 0.

Error Codes

BadDrawable

BadImplementation

Implementation Specifics

This Xlib subroutine is part of AIXwindows Run Time Environment in AIXwindows Environment/6000.

Related Information

The **GetGeometry** protocol request.

XGetIconName Subroutine

Purpose

Gets the name to be displayed for a window icon.

Libraries

Enhanced X-Windows Library (**libX11.a**)

FORTRAN 77 Library (**libXfx.a**)

C Syntax

int XGetIconName(*DisplayPtr*, *WindowID*, *IconNameReturn*)
Display **DisplayPtr*;
Window *WindowID*;
char ***IconNameReturn*;

FORTRAN Syntax

integer*4 fxgeticonname
external fxgeticonname
integer*4 *DisplayPtr*
integer*4 *WindowID*
integer*4 *IconNameReturn*
integer*4 *Reasoncode*
ReasonCode = **fxgeticonname**(*DisplayPtr*, *WindowID*, *IconNameReturn*)

Description

The **XGetIconName** subroutine gets the name to be displayed in the window icon.

If no name is specified for a window icon, the **XGetIconName** subroutine sets the *IconNameReturn* parameter to the value of **NULL.**

Parameters

DisplayPtr	Specifies the connection to the X Server.
IconNameReturn	Returns a pointer to the null-terminated string which represents the window icon name.
WindowID	Specifies the window ID for the target window.

Return Values

Nonzero	Indicates that the **XGetIconName** subroutine executes successfully.
0	Indicates that no window icon name has been specified.

Error Codes

BadWindow

BadImplementation

Implementation Specifics

This Xlib subroutine is part of AIXwindows Run Time Environment in AIXwindows Environment/6000.

Related Information

The **XFree** subroutine.

XGetIconSizes Subroutine

Purpose

Gets the values of the icon sizes.

Libraries

Enhanced X-Windows Library (**libX11.a**)

FORTRAN 77 Library (**libXfx.a**)

C Syntax

Status XGetIconSizes(*DisplayPtr*, *WindowID*, *SizeListReturn*, *CountReturn***)**
Display ***DisplayPtr*;
Window *WindowID*;
XIconSize ****SizeListReturn*;
int ****CountReturn*;

FORTRAN Syntax

integer*4 fxgeticonsizes
external fxgeticonsizes
integer*4 *DisplayPtr*
integer*4 *WindowID*, *SizeListReturn*, *CountReturn*
integer*4 *ReturnCode*
ReturnCode = **fxgeticonsizes(***DisplayPtr*, *WindowID*, *SizeListReturn*, *CountReturn***)**

Description

The **XGetIconSizes** subroutine returns the values of the icon sizes.

The **XGetIconSizes** subroutine should be called by an application to determine the best icon sizes for the window manager. The application should then use the **XSetWMHints** subroutine to supply the window manager with an icon pixmap or window in one of the supported sizes.

Using the **XFree** subroutine frees the data allocated in the *SizeListReturn* parameter.

Parameters

CountReturn	Returns the number of items in the size list.
DisplayPtr	Specifies the connection to the X Server.
SizeListReturn	Returns a pointer to the size list.
WindowID	Specifies the window ID.

Return Values

0	Indicates that the window manager has not set icon sizes.
Nonzero	Indicates that the number of items in the size list.

Error Codes

BadWindow

BadImplementation

Implementation Specifics

This Xlib subroutine is part of AIXwindows Run Time Environment in AIXwindows Environment/6000.

Related Information

The **GetProperty** protocol request.

The **XFree** subroutine, **XSetWMHints** subroutine.

XGetImage Subroutine

Purpose

Gets the contents of a rectangle in a specified drawable.

Libraries

Enhanced X-Windows Library (**libX11.a**)

FORTRAN 77 Library (**libXfx.a**)

C Syntax

XImage *XGetImage(*DisplayPtr, DrawableID, X, Y, Width, Height, PlaneMask, Format***)**
Display * *DisplayPtr*;
Drawable *DrawableID*;
int *X*, *Y*;
unsigned int *Width*, *Height*;
long *PlaneMask*;
int *Format*;

FORTRAN Syntax

integer*4 fxgetimage
external fxgetimage
integer*4 *DisplayPtr*
integer*4 *DrawableID*
integer*4 *X*, *Y*, *Width*, *Height*
integer*4 *PlaneMask*, *Format*
integer*4 *Image*
Image = **fxgetimage(***DisplayPtr*, *DrawableID*, *X*, *Y*, *Width*, *Height*, *PlaneMask*, *Format***)**

Description

The **XGetImage** subroutine returns the contents of the **XImage** data structure for a specified rectangle in a drawable.

If the *Format* parameter is the value of **XYPixmap**, the **XGetImage** subroutine returns only the bit planes specified in the *PlaneMask* parameter. If the *PlaneMask* parameter requests only a subset of the planes of the display, the depth of the returned image will be the number of planes requested.

If the *Format* parameter is the value of **ZPixmap**, the **XGetImage** subroutine returns 0 for the bits in all planes not specified in the *PlaneMask* parameter.

The **XGetImage** subroutine performs no range-checking on the values in the *PlaneMask* parameter and ignores extraneous bits.

The **XGetImage** subroutine returns the depth of the image, as specified when the drawable was created, to the **XImage** data structure. When the Format parameter is the value of **XYPixmap**, the depth is given by the number of bits set to 1 in the *PlaneMask* parameter.

If the drawable is a pixmap, the specified rectangle must be wholly contained within the pixmap, or a **BadMatch** error results.

If the drawable is a window, the window must be viewable. The specified rectangle of the window must be fully viewable on the screen and wholly contained within the outside edges

of the window when there are no inferiors or overlapping windows. Otherwise, a **BadMatch** error results. The borders of the window can be included and read.

If the window has backing-store, the backing-store contents are returned for regions of the window that are obscured by nonferior other windows that are not inferior to it. If the window does not have backing-store, the returned contents of such obscured regions are undefined. The returned contents of visible regions of inferiors of a different depth than the specified window depth are also undefined. The pointer cursor image is not included in the returned contents.

Parameters

DisplayPtr	Specifies the connection to the X Server.
DrawableID	Specifies the drawable.
Format	Specifies the format for the image as the value of **XYPixmap** or **ZPixmap**.
Height	Specifies the height which defines the dimensions of the rectangle of the subimage.
PlaneMask	Specifies the plane mask.
Width	Specifies the width which defines the dimensions of the rectangle of the subimage.
X	Specifies the x coordinate. This coordinate, which is relative to the origin of the drawable, defines the upper-left corner of the rectangle.
Y	Specifies the y coordinate. This coordinate, which is relative to the origin of the drawable, defines the upper-left corner of the rectangle.

Error Codes

BadDrawable

BadImplementation

BadMatch

BadValue

Implementation Specifics

This Xlib subroutine is part of AIXwindows Run Time Environment in AIXwindows Environment/6000.

Related Information

The **XImage** data structure.

The **GetImage** protocol request.

XGetInputFocus Subroutine

Purpose

Gets the current input focus.

Libraries

Enhanced X-Windows Library (**libX11.a**)

FORTRAN 77 Library (**libXfx.a**)

C Syntax

XGetInputFocus(*DisplayPtr*, *FocusReturn*, *RevertToReturn*)
Display **DisplayPtr*;
Window **FocusReturn*;
int **RevertToReturn*;

FORTRAN Syntax

external fxgetinputfocus
integer*4 *DisplayPtr*
integer*4 *FocusReturn, RevertToReturn*
call fxgetinputfocus(*DisplayPtr*, *FocusReturn*, *RevertToReturn*)

Description

The **XGetInputFocus** subroutine returns the focus window ID and the current focus state.

Parameters

DisplayPtr	Specifies the connection to the X Server.
FocusReturn	Returns the focus window ID, which can be:
	PointerRoot
	None
RevertToReturn	Returns the current focus state, which can be:
	RevertToParent
	RevertToPointerRoot
	RevertToNone

Error Code

BadImplementation

Implementation Specifics

This Xlib subroutine is part of AIXwindows Run Time Environment in AIXwindows Environment/6000.

Related Information

The **GetInputFocus** protocol request.

The **XSetInputFocus** subroutine.

XGetKeyboardControl Subroutine

Purpose

Gets the current keyboard settings.

Libraries

Enhanced X-Windows Library (**libX11.a**)

FORTRAN 77 Library (**libXfx.a**)

C Syntax

XGetKeyboardControl(*DisplayPtr*, *ValuesReturn*)
Display ***DisplayPtr**;
XKeyboardState ***ValuesReturn**;

FORTRAN Syntax

external fxgetkeyboardcontrol
integer*4 *DisplayPtr*
integer*4 *ValuesReturn*
call fxgetkeyboardcontrol (*DisplayPtr, ValuesReturn*)

Description

The **XGetKeyboardControl** subroutine returns the current control values for the keyboard to the **XKeyboardState** data structure.

Parameters

DisplayPtr	Specifies the connection to the X Server.
ValuesReturn	Returns the current keyboard parameter in the specified **XKeyboardState** data structure.

Implementation Specifics

This Xlib subroutine is part of AIXwindows Run Time Environment in AIXwindows Environment/6000.

Related Information

The **GetKeyboardControl** protocol request.

The **XKeyboardState** data structure.

XGetKeyboardMapping Subroutine

Purpose

Obtains the symbols for the specified key codes.

Libraries

Enhanced X-Windows Library (**libX11.a**)

FORTRAN 77 Library (**libXfx.a**)

C Syntax

KeySym *XGetKeyboardMapping (*DisplayPtr*, *FirstKeycodeWanted*, *KeycodeCount*, *KeysymPerKeycodeReturn*)
Display **DisplayPtr*;
KeyCode *FirstKeycodeWanted*;
int *KeycodeCount*;
int **KeysymPerKeycodeReturn*;

FORTRAN Syntax

integer*4 fxgetkeyboardmapping
external fxgetkeyboardmapping
integer*4 *DisplayPtr*
integer*4 *FirstKeycodeWanted*
integer*4 *KeycodeCount*
integer*4 *KeysymPerKeycodeReturn*
integer*4 *Keysym*
Keysym = **fxgetkeyboardmapping**(*DisplayPtr*, *FirstKeycodeWanted*, *KeycodeCount*, *KeysymPerKeycodeReturn*)

Description

The **XGetKeyboardMapping** subroutine returns the symbols for the specified number of key codes starting with the value in the *FirstKeycodeWanted* parameter. This value must be greater than or equal to the *min_keycode* as returned by the **XDisplayKeycodes** subroutine, or a **BadValue** error occurs.

In addition, the following expression must be less than or equal to the *max_keycode* as returned by the **XDisplayKeycodes** subroutine:

```
FirstKeycodeWanted + KeycodeCount - 1
```

The number of elements in the key symbols list is:

```
KeycodeCount * KeysymbolsPerKeycodeReturn
```

Then, the key symbol `N` counting from 0 for the `K` key code has the following index in the list, counting from 0:

```
(K - FirstKeycodeWanted) * KeysymbolsPerKeycodeReturn + N
```

Use the **XFree** subroutine to free the storage returned by the **XGetKeyboardMapping** subroutine.

Parameters

DisplayPtr — Specifies the connection to the X Server.

FirstKeycodeWanted	Specifies the first key code to be returned.
KeycodeCount	Specifies the number of key codes to be returned.
KeysymPerKeycodeReturn	Returns the number of key symbols per key code.

This value is chosen arbitrarily by the X 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 key codes.

Error Codes

BadValue

BadImplementation

Implementation Specifics

This Xlib subroutine is part of AIXwindows Run Time Environment in AIXwindows Environment/6000.

Related Information

The **XDisplayKeycodes** subroutine.

The **GetKeyboardMapping** protocol request.

XGetModifierMapping Subroutine

Purpose

Gets the keycodes used as modifiers.

Libraries

Enhanced X-Windows Library (**libX11.a**)

FORTRAN 77 Library (**libXfx.a**)

C Syntax

XModifierKeymap *XGetModifierMapping(*DisplayPtr***);**
Display ***DisplayPtr;*

FORTRAN Syntax

integer*4 fxgetmodifiermapping
external fxgetmodifiermapping
integer*4 *DisplayPtr*
integer*4 *ModifierMap*
ModifierMap = **fxgetmodifiermapping(***DisplayPtr***)**

Description

The **XGetModifierMapping** subroutine returns a pointer to a newly created **XModifierKeymap** data structure that contains the keycodes being used as modifiers. The **XModifierKeymap** data structure should be freed after use with the **XFreeModifierMapping** subroutine.

If only zero values appear in the set for any modifier, that modifier is disabled.

Parameter

DisplayPtr Specifies the connection to the X Server.

Error Code

BadImplementation

Implementation Specifics

This Xlib subroutine is part of AIXwindows Run Time Environment in AIXwindows Environment/6000.

Related Information

The **XModifierKeymap** data structure.

The **GetModifierMapping** protocol request.

XGetMotionEvents Subroutine

Purpose

Gets the motion history of a window for a specified period.

Libraries

Enhanced X-Windows Library (**libX11.a**)

FORTRAN 77 Library (**libXfx.a**)

C Syntax

XTimeCoord *XGetMotionEvents(*DisplayPtr, WindowID, Start,Stop,*
*NumberEventsReturn***)**
Display ***DisplayPtr*;**
Window *WindowID***;**
Time *Start*, *Stop***;**
int ***NumberEventsReturn*;**

FORTRAN Syntax

integer*4 fxgetmotionevents
external fxgetmotionevents
integer*4 *DisplayPtr*
integer*4 *WindowID*, *Start*, *Stop*, *NumberEventsReturn*
integer*4 *TimeCoordinates*
TimeCoordinates **= fxgetmotionevents(***DisplayPtr, WindowID, Start, Stop,*
*NumberEventsReturn***)**

Description

The **XGetMotionEvents** subroutine returns all events in the motion history buffer that fall between specified start and stop times (inclusive), and that have coordinates within the specified window (including borders) at the window's present placement.

The *Start* and *Stop* parameter times are set to in a timestamp, expressed in milliseconds, or as the value of **CurrentTime**.

If the *Start* parameter time is later than the *Stop* parameter time, or if the *Start* parameter time is in the future, no events are returned. If the *Stop* parameter time is in the future, it is equivalent to specifying a value of the **CurrentTime**.

The *X* and *Y* parameter values are set to the coordinates of the pointer and are reported relative to the origin of the specified window.

The return type for the **XGetMotionEvents** subroutine is defined as follows:

```
typedef struct {
        Time time;
        unsigned short x, y;
} XTimeCoord;
```

time Specifies the time in milliseconds.

x Specifies the x coordinate of the pointer relative to the origin of the specified window.

y Specifies the y coordinate of the pointer relative to the origin of the specified window.

Use the **XFree** subroutine to free the data returned.

Parameters

DisplayPtr	Specifies the connection to the X Server.
WindowID	Specifies the window ID of the window for which 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 the value of **CurrentTime**.
NumberEventsReturn	Returns the number of events from the motion history buffer.

Error Codes

BadImplementation

BadWindow

Implementation Specifics

This Xlib subroutine is part of AIXwindows Run Time Environment in AIXwindows Environment/6000.

Related Information

The **XFree** subroutine.

The **GetMotionEvents** protocol.

XGetNormalHints Subroutine

Purpose

Gets the size hints for a window in its normal state.

Libraries

Enhanced X-Windows Library (**libX11.a**)

FORTRAN 77 Library (**libXfx.a**)

C Syntax

Status XGetNormalHints(*DisplayPtr*, *WindowID*, *HintsReturn*);
Display **DisplayPtr*;
Window *WindowID*;
XSizeHints **HintsReturn*;

FORTRAN Syntax

integer*4 fxgetnormalhints
external fxgetnormalhints
integer*4 *DisplayPtr*
integer*4 *WindowID*
integer*4 *HintsReturn*
integer*4 *ReturnCode*
ReturnCode = **fxgetnormalhints(***DisplayPtr*, *WindowID*, *HintsReturn*)

Description

The **XGetNormalHints** subroutine returns the size hints for a window in its normal state.

Parameters

DisplayPtr	Specifies the connection to the X Server.
WindowID	Specifies the window ID.
HintsReturn	Returns the sizing hints for the window in its normal state.

Return Values

0	If the application specified no normal size hints for the specified window.
Nonzero	If the **XGetNormalHints** subroutine succeeds in returning the size hints for a window in its normal state.

Error Codes

BadImplementation

BadWindow

Implementation Specifics

This Xlib subroutine is part of AIXwindows Run Time Environment in AIXwindows Environment/6000.

Related Information

The **GetProperty** protocol.

XGetPixel Subroutine

Purpose

Gets a pixel value in an image.

Libraries

Enhanced X-Windows Library (**libX11.a**)

FORTRAN 77 Library (**libXfx.a**)

C Syntax

unsigned long XGetPixel(*XImagePtr, X, Y***)**
XImage ****XImagePtr*;**
int *X*;
int *Y*;

FORTRAN Syntax

integer*4 fxgetpixel
external fxgetpixel
integer*4 *XImagePtr, X, Y,*
integer*4 *Pixel*
Pixel **= fxgetpixel(***XImage, X, Y***)**

Description

The **XGetPixel** subroutine gets the specified pixel from the named image. The pixel value is returned in normalized format, where the least-significant byte of the long is the least-significant byte of the pixel.

Parameters

XImagePtr	Specifies a pointer to the image.
X	Specifies the x coordinate of the upper left corner relative to the origin of the image.
Y	Specifies the y coordinate of the upper left corner relative to the origin of the image.

Error Code

BadImplementation

Implementation Specifics

This Xlib subroutine is part of AIXwindows Run Time Environment in AIXwindows Environment/6000.

Related Information

The **XImage** data structure.

XGetPointerControl Subroutine

Purpose

Gets the current pointer acceleration parameters.

Libraries

Enhanced X-Windows Library (**libX11.a**)

FORTRAN 77 Library (**libXfx.a**)

C Syntax

XGetPointerControl(*DisplayPtr*, *AccelerationNumeratorReturn*, *AccelerationDenominatorReturn*, *ThresholdReturn*)
Display **DisplayPtr*;
int **AccelerationNumeratorReturn*, **AccelerationDenominatorReturn*;
int **ThresholdReturn*;

FORTRAN Syntax

external fxgetpointercontrol
integer*4 *DisplayPtr*
integer*4 *AccelerationNumeratorReturn*
integer*4 *AccelerationDenominatorReturn*, *ThresholdReturn*
call fxgetpointercontrol(*DisplayPtr*, *AccelerationNumeratorReturn*, *AccelerationDenominatorReturn*, *ThresholdReturn*)

Description

The **XGetPointerControl** subroutine returns the current acceleration multiplier and acceleration threshold of the pointer.

Parameters

AccelerationDenominatorReturn	Returns the denominator for the acceleration multiplier.
AccelerationNumeratorReturn	Returns the numerator for the acceleration multiplier.
DisplayPtr	Specifies the connection to the X Server.
ThresholdReturn	Returns the acceleration threshold.

Error Code

BadImplementation

Implementation Specifics

This Xlib subroutine is part of AIXwindows Run Time Environment in AIXwindows Environment/6000.

Related Information

The **GetPointerControl** protocol request.

The **XChangePointerControl** subroutine.

XGetPointerMapping Subroutine

Purpose

Gets the mapping of the buttons on the pointer.

Libraries

Enhanced X-Windows Library (**libX11.a**)

FORTRAN 77 Library (**libXfx.a**)

C Syntax

int XGetPointerMapping(*DisplayPtr*, *MapReturn*, *NumberMap*)
Display **DisplayPtr*;
unsigned char *MapReturn*[];
int *NumberMap*;

FORTRAN Syntax

integer*4 fxgetpointermapping
external fxgetpointermapping
integer*4 *DisplayPtr*
integer*4 *MapReturn*
integer*4 *NumberMap*
integer*4 *ReturnCode*
ReturnCode = **fxgetpointermapping(***DisplayPtr*, *MapReturn*, *NumberMap***)**

Description

The **XGetPointerMapping** subroutine 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 actual number of physical buttons. The nominal mapping for a pointer is the identity mapping as follows:

```
map[i]=i
```

and only the first *NumberMap* parameter elements are returned in the *MapReturn* parameter.

Parameters

DisplayPtr	Specifies the connection to the X Server.
MapReturn	Specifies the mapping list.
NumberMap	Specifies the number of items in the mapping list.

Error Code

BadImplementation

Implementation Specifics

This Xlib subroutine is part of AIXwindows Run Time Environment in AIXwindows Environment/6000.

Related Information

The **GetPointerMapping** protocol.

XGetScreenSaver Subroutine

Purpose

Gets the current screen saver values.

Libraries

Enhanced X-Windows Library (**libX11.a**)

FORTRAN 77 Library (**libXfx.a**)

C Syntax

XGetScreenSaver(*DisplayPtr, TimeoutReturn, IntervalReturn, PreferBlankingReturn, AllowExposuresReturn*)
Display **DisplayPtr*;
int **TimeoutReturn*, **IntervalReturn*;
int **PreferBlankingReturn*;
int **AllowExposuresReturn*;

FORTRAN Syntax

external fxgetscreensaver
integer*4 *DisplayPtr*
integer*4 *TimeoutReturn, IntervalReturn*
integer*4 *PreferBlankingReturn, AllowExposuresReturn*
call fxgetscreensaver(*DisplayPtr, TimeoutReturn, IntervalReturn, PreferBlankingReturn, AllowExposuresReturn*)

Description

The **XGetScreenSaver** subroutine obtains the current screen saver value.

Parameters

AllowExposuresReturn	Returns the current screen save control value. This parameter can have the following values: **AllowExposures** **DefaultExposures** **DontAllowExposures**
DisplayPtr	Specifies the connection to the X Server.
IntervalReturn	Returns the interval between screen saver invocations.
PreferBlankingReturn	Returns the current screen blanking preference. This parameter can have the following values: **DefaultBlanking** **DontPreferBlanking** **PreferBlanking**
TimeoutReturn	Returns the timeout, in minutes, until the screen saver turns on.

Error Code

BadImplementation

Implementation Specifics

This Xlib subroutine is part of AIXwindows Run Time Environment in AIXwindows Environment/6000.

Related Information

The **XForceScreenSaver** and the **XSetScreenSaver** subroutines.

The **SetScreenSaver** protocol request and the **GetScreenSaver** protocol request.

XGetSelectionOwner Subroutine

Purpose

Gets the selection owner.

Libraries

Enhanced X-Windows Library (**libX11.a**)

FORTRAN 77 Library (**libXfx.a**)

C Syntax

Window XGetSelectionOwner(*DisplayPtr*, *Selection***)**
Display ****DisplayPtr*;**
Atom *Selection*;

FORTRAN Syntax

integer*4 fxgetselectionowner
external fxgetselectionowner
integer*4 *DisplayPtr*
integer*4 *Selection*
integer*4 *ID*
ID **= fxgetselectionowner(***DisplayPtr*, *Selection***)**

Description

The **XGetSelectionOwner** subroutine returns the window ID associated with the window that currently owns the specified selection.

Parameters

DisplayPtr	Specifies the connection to the X Server.
Selection	Specifies the selection atom to be returned.

Return Values

None	Indicates that if no selection is specified or if no owner exists. The window ID associated with the selection is returned.

Error Codes

BadAtom

BadImplementation

Implementation Specifics

This Xlib subroutine is part of AIXwindows Run Time Environment in AIXwindows Environment/6000.

Related Information

The **GetSelectionOwner** protocol.

XGetSizeHints Subroutine

Purpose

Gets the values of type WM_SIZE_HINTS properties.

Libraries

Enhanced X-Windows Library **(libX11.a)**

FORTRAN 77 Library **(libXfx.a)**

C Syntax

Status XGetSizeHints (*DisplayPtr*, *WindowID*, *HintsReturn*, *Property*)
Display **DisplayPtr*;
Window *WindowID*;
XSizeHints **HintsReturn*;
Atom *Property*;

FORTRAN Syntax

integer*4 fxgetsizehints
external fxgetsizehints
integer*4 *DisplayPtr*
integer*4 *WindowID*, *HintsReturn*, *Property*
integer*4 *ReturnCode*
ReturnCode = **fxgetsizehints**(*DisplayPtr*, *WindowID*, *HintsReturn*, *Property*)

Description

The **XGetSizeHints** subroutine returns the **XSizeHints** structure for the named property and the specified window. This subroutine is used by the **XGetNormalHints** subroutine and the **XGetZoomHints** subroutine. The **XSizeHints** structure can also be used to retrieve the value of any WM_SIZE_HINTS properties. Thus, it can be useful if other properties of that type are defined.

Parameters

DisplayPtr	Specifies the connection to the X Server.
WindowID	Specifies the window ID.
HintsReturn	Returns the size hints.
Property	Specifies the property atom.

Return Values

0	If a size hint was undefined.
Nonzero	If a size hint was defined.

Error Codes

BadAtom

BadImplementation

BadWindow

Implementation Specifics

This Xlib subroutine is part of AIXwindows Run Time Environment in AIXwindows Environment/6000.

Related Information

The **XSizeHints** data structure.

The **XGetNormalHints** subroutine and **XGetZoomHints** subroutine.

The **GetProperty** protocol request.

XGetStandardColormap Subroutine

Purpose

Gets the colormap associated with the specified atom.

Libraries

Enhanced X-Windows Library (**libX11.a**)

FORTRAN 77 Library (**libXfx.a**)

C Syntax

Status XGetStandardColormap (*DisplayPtr*, *WindowID*, *ColormapReturn*, *Property*)
Display **DisplayPtr*;
Window *WindowID*;
XStandardColormap **ColormapReturn*;
Atom *Property*;

FORTRAN Syntax

integer*4 fxgetstandardcolormap
external fxgetstandardcolormap
integer*4 *DisplayPtr*
integer*4 *WindowID*, *ColormapReturn*, *Property*
integer*4 *ReturnCode*
ReturnCode = **fxgetstandardcolormap** (*DisplayPtr*, *WindowID*, *ColormapReturn*, *Property*)

Description

The **XGetStandardColormap** subroutine returns the colormap definition associated with the atom supplied as the *Property* parameter.

This colormap can be used to convert RGB values into pixel values.

Using addition rather than the logical OR for composing pixel values permits allocations where the RGB value is not aligned to bit boundaries.

To get the standard gray-scale colormap for a display screen, use the **XGetStandardColormap** subroutine with the following syntax:

```
XGetStandardColormap(display,DefaultRootWindow(display),Cmap,XA_
_RGB_GRAY_MAP);
```

This colormap can be used to convert the RGB values into pixel values. For example, given an **XStandardColormap** subroutine 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;
```

Parameters

ColormapReturn	Returns the colormap associated with the specified atom.
DisplayPtr	Specifies the connection to the X Server.
Property	Specifies the property atom.
WindowID	Specifies the window ID.

Error Codes

BadAtom

BadImplementation

BadWindow

Implementation Specifics

This Xlib subroutine is part of AIXwindows Run Time Environment in AIXwindows Environment/6000.

Related Information

The **XStandardColormap** data structure.

XGetSubImage Subroutine

Purpose

Updates the specified image with the specified subimage.

Library

Enhanced X-Windows Library (**libX11.a**)

FORTRAN 77 Library (**libXfx.a**)

C Syntax

XImage *XGetSubImage(*DisplayPtr, DrawableID, X, Y, Width, Height, PlaneMask, Format,*
*DestinationImage, DestinationX, DestinationY***)**
Display **DisplayPtr*;
Drawable *DrawableID*;
int *X*, *Y*;
unsigned int *Width*, *Height*;
unsigned long *PlaneMask*;
int *Format*;
XImage **DestinationImage*;
int *DestinationX*, *DestinationY*;

FORTRAN Syntax

integer*4 fxgetsubimage
external fxgetsubimage
integer*4 *DisplayPtr*
integer*4 *DrawableID*
integer*4 *X*, *Y*, *Width*, *Height*
integer*4 *PlaneMask*, *Format*
integer*4 *DestinationImage*, *DestinationX*, *DestinationY*
integer*4 *SubImage*
SubImage = **fxgetsubimage(***DisplayPtr, DrawableID, X, Y, Width, Height, PlaneMask,*
*Format, DestinationImage, DestintationX, DestinationY***)**

Description

The **XGetSubImage** subroutine updates the destination image with the specified subimage. If the *Format* parameter is **XYPixmap**, the image contains only the bit planes passed to the *PlaneMask* parameter. If the *Format* parameter is **ZPixmap**, this subroutine returns as zero the bits in all planes not specified in the *PlaneMask* parameter.

The **XGetSubImage** subroutine performs no range checking on the values in the *PlaneMask* parameter and ignores extraneous bits. As a convenience, this subroutine returns a pointer to the same **XImage** structure specified by *DestinationImage*.

The depth of the destination **Ximage** structure 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.

If the drawable is a pixmap, the given rectangle must be wholly contained within the pixmap, or a **BadMatch** error results. If the drawable is a window, the window must be viewable, and it must also be the case that if there are no inferiors or overlapping windows, the specified rectangle of the window is fully visible on the screen. Otherwise, a **BadMatch** error results.

If the window has backing-store, the backing-store contents are returned for regions of the window that are obscured by noninferior windows. Otherwise, the contents returned for obscured regions of the window are undefined.

The contents returned for visible regions of inferiors with a depth different than the depth of the specified window are also undefined.

Parameters

DisplayPtr	Specifies the connection to the X Server.
DrawableID	Specifies the drawable.
X	Specifies the x coordinate relative to the origin of the specified drawable. The x and y coordinates define the upper-left corner of the rectangle.
Y	Specifies the y coordinate relative to the origin of the specified drawable. The x and y coordinates define the upper-left corner of the rectangle.
Width	Specifies the width of the rectangle of the subimage.
Height	Specifies the height of the rectangle of the subimage.
PlaneMask	Specifies the plane mask.
Format	Specifies the format for the image. The *Format* parameter can have the following values: **XYBitmap** **XYPixmap** **ZPixmap**
DestinationImage	Specifies the destination image.
DestinationX	Specifies the x coordinate of the destination rectangle. The x and y coordinates, specify the upper-left corner of the destination rectangle relative to its origin. These coordinates determine where the subimage is placed within the destination image.
DestinationY	Specifies the y coordinate of the destination rectangle. The x and y coordinates, specify the upper-left corner of the destination rectangle relative to its origin. These coordinates determine where the subimage is placed within the destination image.

Error Codes

BadDrawable

BadGC

BadImplementation

BadMatch

BadValue

Implementation Specifics

This Xlib subroutine is part of AIXwindows Run Time Environment in AIXwindows Environment/6000.

Related Information

The **XImage** data structure.

XGetTransientForHint Subroutine

Purpose

Gets WM_TRANSIENT_FOR property for a window.

Libraries

Enhanced X-Windows Library (**libX11.a**)

FORTRAN 77 Library (**libXfx.a**)

C Syntax

Status XGetTransientForHint(*DisplayPtr*, *WindowID*, *PropertyWindowReturn***)**
Display ******DisplayPtr*;
Window *WindowID*;
Window ******PropertyWindowReturn*;

FORTRAN Syntax

integer*4 fxgettransientforhint
external fxgettransientforhint
integer*4 *DisplayPtr*
integer*4 *WindowID*, *PropertyWindowReturn*
integer*4 *ReturnCode*
ReturnCode = **fxgettransientforhint(***DisplayPtr, WindowID*, *PropertyWindowReturn***)**

Description

The **XGetTransientForHint** subroutine obtains the WM_TRANSIENT_FOR property for the specified window.

Parameters

DisplayPtr	Specifies the connection to the X Server.
WindowID	Specifies the window ID.
PropertyWindowReturn	Returns the WM_TRANSIENT_FOR property of the specified window.

Return Values

False	The **XGetTransientForHint** subroutine is not successful.
True	The **XGetTransientForHint** subroutine is successful.

Error Codes

BadImplementation

BadWindow

Implementation Specifics

This Xlib subroutine is part of AIXwindows Run Time Environment in AIXwindows Environment/6000.

XGetVisualInfo Subroutine

Purpose

Gets list of visual information structures.

Libraries

Enhanced X-Windows Library (**libX11.a**)

FORTRAN 77 Library (**libXfx.a**)

C Syntax

XVisualInfo *XGetVisualInfo(*DisplayPtr*, *VisualInformationMask*,
VisualInformationTemplate, *NumberItemsReturn***)**
Display **DisplayPtr*;
long *VisualInformationMask*;
XVisualInfo **VisualInformationTemplate*;
int **NumberItemsReturn*;

FORTRAN Syntax

integer*4 fxgetvisualinfo
external fxgetvisualinfo
integer*4 *DisplayPtr*, *VisualInformationMask*
integer*4 *VisualInformationTemplate*
integer*4 *NumberItemsReturn*
integer*4 *VisualInformation*
VisualInformation = **fxgetvisualinfo(***DisplayPtr*, *VisualInformationMask*,
VisualInformationTemplate,
*NumberItemsReturn***)**

Description

The **XGetVisualInfo** subroutine gets a list of visual structures that match the attributes specified in the *VisualInformationTemplate* parameter. If visual structures match the template, this subroutine returns a pointer to the list of visual structures. To free the data returned by this function, use the **XFree** subroutine.

Parameters

DisplayPtr	Specifies the connection to the X Server.
NumberItemsReturn	Returns the number of matching visual structures.
VisualInformationMask	Specifies the visual mask value.
VisualInformationTemplate	Specifies the visual attributes for matching the visual structures. See Determining the Appropriate Visual.

Return Values

NULL	No visual structures match the template
Visual structures	The array that matches the template

Error Code

BadImplementation

Implementation Specifics

This Xlib subroutine is part of AIXwindows Run Time Environment in AIXwindows Environment/6000.

Related Information

The **XVisualInfo** data structure.

Determining the Appropriate Visual and the **XFree** subroutine.

XGetWindowAttributes Subroutine

Purpose

Gets the current attributes for the specified window.

Libraries

Enhanced X-Windows Library (**libX11.a**)

FORTRAN 77 Library (**libXfx.a**)

C Syntax

Status XGetWindowAttributes(*DisplayPtr*, *WindowID*, *WindowAttributesReturn***)**
**Display ****DisplayPtr*;
Window *WindowID*;
**XWindowAttributes ****WindowAttributesReturn*;

FORTRAN Syntax

integer*4 fxgetwindowattributes
external fxgetwindowattributes
integer*4 *DisplayPtr*
integer*4 *WindowID*
integer*4 *WindowAttributesReturn*
integer*4 *Status*
Status = **fxgetwindowattributes(***DisplayPtr*, *WindowID*, *WindowAttributesReturn***)**

Description

The **XGetWindowAttributes** subroutine obtains the current attributes for a specified window. This subroutine returns the current attributes to an **XWindowAttributes** data structure.

Parameters

DisplayPtr	Specifies the connection to the X Server.
WindowID	Specifies the window ID for the window for which current attributes are to be obtained.
WindowAttributesReturn	Returns the attributes in the **XWindowAttributes** subroutine structure of the specified window.

Return Values

False	The **XGetWindowAttributes** subroutine is not successful. None of the return arguments are updated.
True	The **XGetWindowAttributes** subroutine is successful.

Error Codes

BadDrawable

BadImplementation

BadWindow

Implementation Specifics

This Xlib subroutine is part of AIXwindows Run Time Environment in AIXwindows Environment/6000.

Related Information

The **XWindowAttributes** data structure.

The **GetWindowAttributes** protocol request, **GetGeometry** protocol request.

XGetWindowProperty Subroutine

Purpose

Gets the atom type and property format for a window.

Libraries

Enhanced X-Windows Library (**libX11.a**)

FORTRAN 77 Library (**libXfx.a**)

C Syntax

int XGetWindowProperty(*DisplayPtr*, *WindowID*, *Property*, ,*LongOffset*,*LongLength*,
Delete, *RequestedType*, *ActualTypeReturn*,
ActualFormatReturn,*NumberItemsReturn*,
BytesAfterReturn,*PropertyReturn*)
Display **DisplayPtr*;
Window *WindowID*;
Atom *Property*;
long *LongOffset*, *LongLength*;
Bool *Delete*;
Atom *RequestedType*;
Atom **ActualTypeReturn*;
int **ActualFormatReturn*;
unsigned long **NumberItemsReturn*;
unsigned long **BytesAfterReturn*;
unsigned char **PropertyReturn*;

FORTRAN Syntax

integer*4 fxgetwindowproperty
external fxgetwindowproperty
integer*4 *DisplayPtr*
integer*4 *WindowID*
integer*4 *Property*
integer*4 *LongOffset*, *LongLength*
integer*4 *Delete*, *RequestedType*, *ActualTypeReturn*
integer*4 *ActualFormatReturn*, *NumberItemsReturn*
integer*4 *BytesAfterReturn*, *PropertyReturn*
integer*4 *ReturnCode*
ReturnCode = **fxgetwindowproperty(***DisplayPtr*, *WindowID*, *Property*, *LongOffset*,
LongLength, *Delete*, *RequestedType*,
ActualTypeReturn, *ActualFormatReturn*,
NumberItemsReturn, *BytesAfterReturn*,
PropertyReturn)

Description

The **XGetWindowProperty** subroutine obtains the atom type and property format for a specified window. This subroutine sets the return parameters according to the following:

- If the specified property does not exist for the specified window, the **XGetWindowProperty** subroutine returns a value of **None** to the *ActualTypeReturn* parameter and a value of 0 to the *ActualFormatReturn* and *BytesAfterReturn* parameters. The *NumberItemsReturn* parameter is empty. The *Delete* parameter is ignored.

- If the specified property exists, but the property type does not match the specified type, the **XGetWindowProperty** subroutine returns the actual property type to the *ActualTypeReturn* parameter. It returns the actual property format (never a value of 0) to the *ActualFormatReturn* parameter. It also returns the property length in bytes (even if the *ActualFormatReturn* parameter is 16-bit or 32-bit) to the *BytesAfterReturn* parameter. It ignores the *Delete* parameter. The *NumberItemsReturn* parameter is empty.
- If the specified property exists and the *RequestedType* parameter is set to the **AnyPropertyType** identifier or if the specified type matches the actual property type, the **XGetWindowProperty** subroutine returns the actual property type to the *ActualTypeReturn* parameter and returns the actual property format (never a value of 0zero) to the *ActualFormatReturn* parameter. The **XGetWindowProperty** subroutine also returns a value to the *BytesAfterReturn* and *NumberItemsReturn* parameters 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 the *LongOffset* parameter makes L negative, an error is generated.

  The *BytesAfterReturn* parameter is A, giving the number of trailing unread bytes in the stored property.

The **XGetWindowProperty** subroutine always allocates one extra byte in the *PropertyReturn* parameter (even if the property is 0 length) and sets it to the value of **ASCII NULL** so that simple properties consisting of characters do not have to be copied into yet another string before use. If the *Delete* parameter is a **True** value and the *BytesAfterReturn* parameter is a value of 0, the subroutine deletes the property from the window and generates a **PropertyNotify** event value on the window.

This subroutine returns a **Success** value if it executes successfully. To free the resulting data, use the **XFree** subroutine.

Parameters

ActualFormatReturn	Returns the actual format (in 8-bit, 16-bit, or 32-bit) of the property.
ActualTypeReturn	Returns the atom identifier that defines the type of the property.
BytesAfterReturn	Returns the number of bytes remaining in the property if a partial read was performed.
Delete	Specifies a Boolean value that determines if the property is to be deleted from the window. It can be set to a **True** or **False** value.
DisplayPtr	Specifies the connection to the X Server.
LongLength	Specifies the length (in 32-bit multiples) of the data to be retrieved.

LongOffset	Specifies the offset (in 32-bit quantities) in the specified property where data will be retrieved.
NumberItemsReturn	Returns the actual number of items transferred.
Property	Specifies the property atom.
PropertyReturn	Returns a pointer to the data in the specified format.
RequestedType	Specifies an atom identifier or the **AnyPropertyType** identifier associated with the property type.
WindowID	Specifies the window ID for the window for which atom type and property format is to be obtained.

Return Value

Success	The **XGetWindowProperty** subroutine runs successfully.

Error Codes

BadAtom

BadImplementation

BadValue

BadWindow

Implementation Specifics

This Xlib subroutine is part of AIXwindows Run Time Environment in AIXwindows Environment/6000.

Related Information

The **XFree** subroutine.

The **GetProperty** protocol request.

XGetWMHints Subroutine

Purpose

Gets the value of the window manager hints atom.

Libraries

Enhanced X-Windows Library (**libX11.a**)

FORTRAN 77 Library (**libXfx.a**)

C Syntax

XWMHints *XGetWMHints(*DisplayPtr***,** *WindowID***)**
Display ***DisplayPtr;*
Window *WindowID*;

FORTRAN Syntax

integer*4 fxgetwmhints
external fxgetwmhints
integer*4 *DisplayPtr*
integer*4 *WindowID*
integer*4 *WMHints*
WMHints **= fxgetwmhints(***DisplayPtr***,** *WindowID***)**

Description

The **XGetWMHints** subroutine reads the value of the window manager hints atom. If successful, this subroutine returns a pointer to a **XWMHints** structure. When finished with the data, free the space used for it by calling the **XFree** subroutine.

Parameters

DisplayPtr Specifies the connection to the X Server.

WindowID Specifies the window ID.

Return Value

NULL The **XGetWMHints** subroutine is unsuccessful. This subroutine is unsuccessful if a WM_HINTS property was set for the specified window.

Pointer to the **XWMHints** structure.

Error Codes

BadImplementation

BadWindow

Implementation Specifics

This Xlib subroutine is part of AIXwindows Run Time Environment in AIXwindows Environment/6000.

Related Information

The **XWMHints** data structure.

The **XFree** subroutine.

The **GetProperty** protocol request.

XGetZoomHints Subroutine

Purpose

Gets values of the zoom hints atom.

Libraries

Enhanced X-Windows Library (**libX11.a**)

FORTRAN 77 Library (**libXfx.a**)

C Syntax

Status XGetZoomHints(*DisplayPtr*, *WindowID*, *ZoomHintsReturn***)**
**Display ****DisplayPtr*;
Window *WindowID*;
**XSizeHints ****ZoomHintsReturn*;

FORTRAN Syntax

integer*4 fxgetzoomhints
external fxgetzoomhints
integer*4 *DisplayPtr*
integer*4 *WindowID*, *ZoomHintsReturn*
integer*4 *Status*
Status = **fxgetzoomhints(***DisplayPtr*, *WindowID*, *ZoomHintsReturn***)**

Description

The **XGetZoomHints** subroutine returns the size hints for a window in its zoomed state. This subroutine returns these hints in its last argument.

The **XGetZoomHints** subroutine can be unsuccessful if the application did not specify the zoom size hints for this window.

Parameters

DisplayPtr	Specifies the connection to the X Server.
WindowID	Specifies the window ID.
ZoomHintsReturn	Returns a pointer to the structure containing the zoom hints.

Return Values

False	The application specified no zoom size hints.
True	The **XGetZoomHints** subroutine succeeds.

Error Codes

BadImplementation

BadWindow

Implementation Specifics

This Xlib subroutine is part of AIXwindows Run Time Environment in AIXwindows Environment/6000.

Related Information

The **XSizeHints** data structure.

The **GetProperty** protocol request.

XGrabButton Subroutine

Purpose

Grabs a mouse button.

Libraries

Enhanced X-Windows Library (**libX11.a**)

FORTRAN 77 Library (**libXfx.a**)

C Syntax

XGrabButton(*DisplayPtr*, *Button*, *Modifiers*, *GrabWindow*, *OwnerEvents*, *EventMask*, *PointerMode*, *KeyboardMode*, *ConfineTo*, *Cursor*)
Display **DisplayPtr*;
unsigned int *Button*;
unsigned int *Modifiers*;
Window *GrabWindow*;
Bool *OwnerEvents*;
unsigned int *EventMask*;
int *PointerMode*, *KeyboardMode*;
Window *ConfineTo*;
Cursor *Cursor*;

FORTRAN Syntax

external fxgrabbutton
integer*4 *DisplayPtr*
integer*4 *ButtonGrab*, *Modifiers*, *GrabWindow*
integer*4 *OwnerEvents*, *EventMask*, *PointerMode*
integer*4 *KeyboardMode*, *ConfineTo*, *Cursor*
call fxgrabbutton(*DisplayPtr*, *ButtonGrab*, *Modifiers*, *GrabWindow*, *OwnerEvents*, *EventMask*, *PointerMode*, *KeyboardMode*, *ConfineTo*, *Cursor*)

Description

The **XGrabButton** subroutine establishes a passive grab procedure. Consequently, in the future, the pointer is actively grabbed (as for the **XGrabPointer** subroutine), the last-pointer-grab time is set to the time at which the button was pressed (as transmitted in the **ButtonPress** event), and the **ButtonPress** event is reported if all of the following conditions are true:

- 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).
- The *GrabWindow* parameter contains the pointer.
- The *ConfineTo* parameter window, if any, is viewable.
- The passive grab on the same button-key combination does not exist on any ancestor of the grab window

The interpretation of the remaining parameters is as for the **XGrabPointer** subroutine. The active grab is ended automatically when the logical state of the pointer has all buttons released (independent of the state of the logical modifier keys).

Note that the logical state of a device (as seen by client applications) can lag the physical state if device event processing is frozen.

This request overrides all previous grab procedures by the same client on the same button-key combinations on the same window. A modifier of the **AnyModifier** value is equivalent to issuing the grab request for all possible modifier combinations (including the combination of no modifiers). It is not required that all modifiers specified have currently assigned key codes. A button of the **AnyButton** value 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.

If some other client has already issued the **XGrabButton** subroutine with the same button/key combination on the same window, a **BadAccess** error results. When using the **AnyModifier** or **AnyButton** value, the request is unsuccessful, and a **BadAccess** error code results (no grabs are established) if there is a conflicting grab procedure for any combination.

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 key codes. A button with the **AnyButton** value 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.

This variable can also be set to the value of **AnyModifier**, which is equivalent to issuing the grab request for all possible modifier combinations (including the combination of no modifiers).

Both the *PointerMode* and *KeyboardMode* parameters can be the **GrabModeSync** or **GrabModeAsync** value.

The **XGrabButton** subroutine is unsuccessful and generates an error if another client issues this subroutine with the same button-key combination on the same window.

The **XGrabButton** subroutine is unsuccessful, does not establish a grab procedure, and generates an error if the **AnyModifier** or **AnyButton** modifier is used with a conflicting grab procedure.

The **XGrabButton** subroutine has no effect on an active grab procedure.

Parameters

Button — Specifies the pointer button to be grabbed when the specified modifier keys are down or the value of **AnyButton** is the modifier.

ConfineTo — Specifies the window ID in which to confine the pointer. Or, this parameter specifies the **None** value if the pointer is not to be confined.

Cursor — Specifies the cursor to be displayed during the grab procedure. Or, this parameter specifies the **None** value if the cursor is not to be displayed.

DisplayPtr — Specifies the connection to the X Server.

EventMask — Specifies what pointer events are reported to the client. This parameter is a bitwise-inclusive OR of the valid pointer event mask bits. It can be set to one of the following values:

ButtonPressMask	**ButtonReleaseMask**
EnterWindowMask	**LeaveWindowMask**
PointerMotionMask	**PointerMotionHintMask**

ButtonMotionMask **KeymapStateMask**

ButtonMotion1Mask **ButtonMotion2Mask**

ButtonMotion3Mask **ButtonMotion4Mask**

ButtonMotion5Mask

GrabWindow Specifies the window ID of the window to be grabbed.

KeyboardMode Controls further processing of keyboard events. This parameter can be the following values:

GrabModeSync **GrabModeAsync**

Modifier Specifies the set of keymasks or the value of **AnyModifier**. This parameter is the bitwise-inclusive OR of valid keymask bits. It can be set to the following valid values:

ShiftMask **LockMask**

Mod1Mask **Mod2Mask**

Mod3Mask **Mod4Mask**

Mod5Mask

OwnerEvents Specifies a Boolean value that indicates whether the pointer events are to be reported as usual or reported with respect to the grab window if selected by the event mask. The *OwnerEvents* parameter can be the following values:

True A key event is reported to this client normally.

False All key events are reported with respect to the *GrabWindow* parameter.

PointerMode Controls further processing of pointer events. This parameter can be the following:

GrabModeSync **GrabModeAsync**

Error Codes

BadAccess

BadCursor

BadImplementation

BadValue

BadWindow

Implementation Specifics

This Xlib subroutine is part of AIXwindows Run Time Environment in AIXwindows Environment/6000.

Related Information

The **XGrabPointer** subroutine.

The **GrabButton** protocol request.

XGrabKey Subroutine

Purpose

Grabs a single key of the keyboard.

Libraries

Enhanced X-Windows Library (**libX11.a**)

FORTRAN 77 Library (**libXfx.a**)

C Syntax

XGrabKey(*DisplayPtr*, *Keycode*, *Modifiers* , *GrabWindow*,*OwnerEvents*, *PointerMode*, *KeyboardMode*)
Display **DisplayPtr*;
int *Keycode*;
unsigned int *Modifiers*;
Window *GrabWindow*;
Bool *OwnerEvents*;
int *PointerMode*, *KeyboardMode*;

FORTRAN Syntax

external fxgrabkey
integer*4 *DisplayPtr*
integer*4 *Keycode*, *Modifiers*, *GrabWindow*
integer*4 *OwnerEvents*, *PointerMode*, *KeyboardMode*
call fxgrabkey(*DisplayPtr*, *Keycode*, *Modifiers*, *GrabWindow*, *OwnerEvents*, *PointerMode*, *KeyboardMode*)

Description

The **XGrabKey** subroutine establishes a passive grab on the keyboard. In the future, the keyboard is actively grabbed (as for the **XGrabKeyboard** subroutine), 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 if the following conditions are true:

- The Keyboard is not grabbed and the specified key, which can be a modifier key, is logically pressed when the specified modifier keys are logically down, and, no other keys are logically down.
- Either the *GrabWindow* parameter is an ancestor of or is the focus window or, the *GrabWindow* parameter is a descendent of the focus window and, it contains the pointer.
- A passive grab on the same key combination does not exist on any ancestor of the *GrabWindow* parameter.

The active grab is terminated automatically when the specified key has been released (independent of the state of the modifier keys).

Note that the logical state of a device (as seen by client applications) can lag the physical state if device event processing is frozen.

The *Modifier* parameter can be set to the modifier **AnyModifier**, which is equivalent to issuing the grab key request for all possible modifier combinations (including no modifiers). It is not required that all modifiers specified have currently assigned KeyCodes. Specifying the

AnyKey modifier is equivalent to issuing the request for all possible KeyCodes. Otherwise, the key must be in the range specified by the *MinimumKeycode* and *MaximumKeycode* parameters in the connection setup or a **BadValue** error results.

Both the *PointerMode* and the *KeyboardMode* parameters be set to the **GrabModeSync** or the **GrabModeAsync** value.

The **XGrabKey** subroutine is unsuccessful and an error is generated if another client issues this subroutine with the same key combination on the same window.

When using the **AnyModifier** or **AnyKey** modifier, the **XGrabKey** subroutine is unsuccessful, no grabs are established, and an error is generated if another grab conflicts with this grab.

Parameters

DisplayPtr Specifies the connection to the X Server.

GrabWindow Specifies the window ID of the window associated with the keys to be grabbed.

Keycode Specifies the keycode or the **AnyKey** modifier to the specific key to be grabbed.

KeyboardMode Controls further processing of keyboard events. This parameter can have the following values:

GrabModeAsync	Indicates that pointer event processing is unaffected by the activation of the grab.
GrabModeSync	Indicates that the state of the pointer as seen by clients appears to freeze, and the X Server generates no further pointer events until the grabbing client issues a releasing **XAllowEvents** subroutine or until the keyboard grab is released. Actual keyboard changes are not lost while the keyboard is frozen; they are queued in the server for later processing.

Modifiers Specifies the set of keymasks. This is a set of bitwise inclusive OR of valid keymask bits. This parameter can have the following values:

ShiftMask	**LockMask**
Mod1Mask	**Mod2Mask**
Mod3Mask	**Mod4Mask**
Mod5Mask	

OwnerEvents Specifies a Boolean value. This parameter can have the following values:

False	Indicates that the pointer events are reported with respect to the *GrabWindow* parameter.

	True	Indicates that a generated key event is reported to the client, if it would normally be reported. Otherwise, the key event is reported with respect to the *GrabWindow* parameter.
PointerMode	Controls further processing of pointer events. This parameter can have the following values:	
	GrabModeAsync	Indicates that pointer event processing is unaffected by the activation of the grab.
	GrabModeSync	Indicates that the state of the pointer as seen by clients appears to freeze, and the X Server generates no further pointer events until the grabbing client issues a releasing **XAllowEvents** subroutine or until the keyboard grab is released. Actual keyboard changes are not lost while the keyboard is frozen; they are queued in the server for later processing.

Error Codes

BadAccess

BadImplementation

BadValue

BadWindow

Implementation Specifics

This Xlib subroutine is part of AIXwindows Run Time Environment in AIXwindows Environment/6000.

Related Information

The **GrabKey** protocol request.

XGrabKeyboard Subroutine

Purpose

Grabs the keyboard.

Libraries

Enhanced X-Windows Library (**libX11.a**)

FORTRAN 77 Library (**libXfx.a**)

C Syntax

int XGrabKeyboard(*DisplayPtr*, *GrabWindow*, *OwnerEvents*, *PointerMode*, *KeyboardMode*, *TimeStamp*)
Display **DisplayPtr*;
Window *GrabWindow*;
Bool *OwnerEvents*;
int *PointerMode*, *KeyboardMode*;
Time *TimeStamp*;

FORTRAN Syntax

integer*4 fxgrabkeyboard
external fxgrabkeyboard
integer*4 *DisplayPtr*
integer*4 *GrabWindow*, *OwnerEvents*
integer*4 *PointerMode*, *KeyboardMode*
integer*4 *TimeStamp*
integer*4 *ReturnCode*
rc = **fxgrabkeyboard**(*DisplayPtr*, *GrabWindow*, *OwnerEvents*, *PointerMode*, *KeyboardMode*, *TimeStamp*)

Description

The **XGrabKeyboard** subroutine actively grabs control of the main keyboard and generates the **FocusIn** and **FocusOut** events. Further key events are reported only to the grabbing client. This subroutine overrides any active keyboard grab by this client.

Both the **KeyPress** and **KeyRelease** events are always reported, independent of any event selection made by the client.

Parameters

DisplayPtr	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. If the *OwnerEvents* parameter is the following:

False	All generated key events are reported with respect to the *GrabWindow* parameter.
True	The generated key event is reported normally to this client.

PointerMode Controls further processing of pointer events. This can be the following:

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 the **XAllowEvents** subroutine call or until the keyboard grab is released.

KeyboardMode Controls further processing of keyboard events. If the *KeyboardMode* parameter is the following:

GrabModeAsync The processing of keyboard events continues normally. If the keyboard is currently frozen by this client, the processing of keyboard events is resumed.

GrabModeSync The keyboard events, as seen by client applications, appear to freeze, and no further keyboard events are generated by the server until the grabbing client issues the **XAllowEvents** subroutine call or until the keyboard is released. Actual keyboard changes are not lost while the keyboard is frozen; instead, these changes are queued for later processing.

TimeStamp Specifies the time in a timestamp, which is expressed in milliseconds, or the **CurrentTime** value.

Return Values

If the **XGrabKeyboard** subroutine is unsuccessful, it returns one of the following values:

AlreadyGrabbed The keyboard is actively grabbed by another client.

GrabNotViewable *The GrabWindow* parameter is not viewable.

GrabInvalidTime 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 the **CurrentTime** value is replaced by the current X Server time.

GrabFrozen The keyboard is frozen by an active grab of another client.

Error Codes

BadImplementation

BadValue

BadWindow

Implementation Specifics

This Xlib subroutine is part of AIXwindows Run Time Environment in AIXwindows Environment/6000.

Related Information

The **XAllowEvents** subroutine.

The **GrabKeyboard** protocol request.

XGrabPointer Subroutine

Purpose

Grabs the pointer.

Libraries

Enhanced X-Windows Library (**libX11.a**)

FORTRAN 77 Library (**libXfx.a**)

C Syntax

int XGrabPointer(*DisplayPtr*, *GrabWindow*, *OwnerEvents*, *EventMask*, *PointerMode*, *KeyboardMode*, *ConfineTo*, *Cursor*, *TimeStamp*)
Display **DisplayPtr*;
Window *GrabWindow*;
Bool *OwnerEvents*;
unsigned int *EventMask*;
int *PointerMode*, *KeyboardMode*;
Window *ConfineTo*;
Cursor *Cursor*;
Time *TimeStamp*;

FORTRAN Syntax

integer*4 fxgrabpointer
external fxgrabpointer
integer*4 *DisplayPtr*
integer*4 *GrabWindow*, *OwnerEvents*, *EventMask*
integer*4 *PointerMode*, *KeyboardMode*
integer*4 *ConfineTo*, *Cursor*, *TimeStamp*
integer*4 *ReturnCode*
ReturnCode = **fxgrabpointer(***DisplayPtr*, *GrabWindow*, *OwnerEvents*, *EventMask*, *PointerMode*, *KeyboardMode*, *ConfineTo*, *Cursor*, *TimeStamp*)

Description

The **XGrabPointer** subroutine actively grabs control of the pointer and returns the **GrabSuccess** value 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.

The **XGrabPointer** subroutine generates the **EnterNotify** and **LeaveNotify** events.

Parameters

ConfineTo — Specifies the window in which to confine the pointer or the value of **None** if the pointer is not to be confined. If the pointer is not in the *ConfineTo* window initially, it is warped automatically to the closest edge of the window before the grab activates. Enter or leave events are generated normally. If the *ConfineTo* window is reconfigured consequently, the pointer is warped automatically to contain it within the window.

Cursor — Specifies the cursor to be displayed during the grab. If the *Cursor* parameter is the value of **NONE**, the normal cursor for that window

is displayed when the pointer is in the *GrabWindow* parameter or one of its subwindows.

Display Specifies the connection to the X Server.

EventMask Specifies which pointer events are reported to the client. The mask is the bitwise inclusive OR of valid pointer event mask bits.

GrabWindow Specifies the ID of the window relative to which events are reported while it is grabbed.

KeyboardMode Controls further processing of keyboard events. The *KeyboardMode* parameter can be the following values:

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 the **XAllowEvents** subroutine 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.

OwnerEvents Specifies a Boolean value that indicates whether the pointer events are to be reported normally or with respect to the grab window if selected by the event mask. The *OwnerEvent* parameter can be the following values:

False All generated pointer events are reported with respect to the *GrabWindow* parameter. These events are reported only if selected by the *EventMask* parameter. Unreported events are discarded.

True It is reported normally if a generated pointer event would be reported to this client normally. Otherwise, the event is reported with respect to the *GrabWindow* parameter and is reported only if selected by the *EventMask* parameter. Unreported events are discarded.

PointerMode Controls further processing of pointer events. The *PointerMode* parameter can be the following values:

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 the **XAllowEvents** subroutine 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.

Time — Specifies the time in a timestamp, which is expressed in milliseconds, or the **CurrentTime** value. The *Time* parameter helps avoid certain situations that can occur. For example, if two applications that normally grab the pointer when clicked on have a specified timestamp, the second application can grab the pointer successfully, while the first application is notified that the pointer was grabbed before its request was processed.

Return Values

If the **XGrabPointer** subroutine is unsuccessful, it returns one of the following values:

GrabNotViewable — The *GrabWindow* or *ConfineTo* window is not viewable.

AlreadyGrabbed — The pointer is actively grabbed by another client.

GrabFrozen — The keyboard is frozen by an active grab of another client.

GrabInvalidTime — 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 the **CurrentTime** value is replaced by the current X Server time.

Error Codes

BadCursor

BadImplementation

BadValue

BadWindow

Implementation Specifics

This Xlib subroutine is part of AIXwindows Run Time Environment in AIXwindows Environment/6000.

Related Information

The **XAllowEvents** subroutine.

The **GrabPointer** protocol request.

XGrabServer Subroutine

Purpose

Grabs the server.

Libraries

Enhanced X-Windows Library (**libX11.a**)

FORTRAN 77 Library (**libXfx.a**)

C Syntax

XGrabServer(*DisplayPtr*)
Display **DisplayPtr*;

FORTRAN Syntax

external fxgrabserver
integer*4 *DisplayPtr*
call fxgrabserver(*DisplayPtr*)

Description

The **XGrabServer** subroutine disables processing of requests and closes down all connections except the one that it arrived on. This subroutine performs a closedown. The X Server should be grabbed only when absolutely necessary because no processing of requests or closedowns on any other connections occur while it is grabbed.

Parameter

DisplayPtr Specifies the connection to the X Server.

Error Code

BadImplementation

Implementation Specifics

This Xlib subroutine is part of AIXwindows Run Time Environment in AIXwindows Environment/6000.

Related Information

The **GrabServer** protocol request.

XIfEvent Subroutine

Purpose

Checks event queue for specified event and removes it.

Libraries

Enhanced X-Windows Library (**libX11.a**)

FORTRAN 77 Library (**libXfx.a**)

C Syntax

XIfEvent(*DisplayPtr*, *EventReturn*, *Predicate*, *Argument*)
DisplayPtr **DisplayPtr*;
XEvent **EventReturn*;
Bool (**Predicate*)();
char **Argument*;

FORTRAN Syntax

external fxifevent
integer*4 *DisplayPtr*
integer*4 *EventReturn*
integer*4 *Predicate*
character*256 *Argument*
call fxifevent(*DisplayPtr*, *EventReturn*, *Predicate*, *Argument*)

The **fxifevent** statement 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 *DisplayPtr*
integer*4 *Event*
integer*4 *Arguments*
call *FunctionName* (*DisplayPtr*, *Event*, *Arguments*)

Description

The **XIfEvent** subroutine checks the event queue for a matching event, using the predicate procedure. If a matching event is found, the **XIfEvent** subroutine removes the event from the queue and copies the event structure into the client-supplied **XEvent** structure. This subroutine flushes the output buffer if it blocks waiting for additional events.

The **XIfEvent** subroutine completes only when the specified predicate procedure returns the value of **True** for an event, which 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.

The **XIfEvent** statement requires a predicate procedure to pass as a parameter. The procedure determines if the event matches the one specified in the corresponding function. The C Syntax for the predicate procedure is as follows:

Bool (**Predicate*)(*DisplayPtr, Event, Argument*)
DisplayPtr **DisplayPtr*
XEvent **Event*
Char **Argument*

- The *DisplayPtr* parameter specifies the connection to the X Server.
- The *Event* parameter specifies a pointer to the **XEvent** structure.
- The *Argument* parameter specifies the argument passed in from the **XIfEvent** subroutine.

The predicate procedure is called once for each event in the queue until it finds a match. After finding a match, the predicate procedure must return the value of **True**. If it does not find a match, it must return the value of **False**. The predicate procedure must decide only if the event is useful and must not call **Xlib** functions.

Parameters

Argument	Specifies the user-supplied argument passed to the predicate procedure.
DisplayPtr	Specifies the connection to the X Server.
EventReturn	Copies the structure of the matched event into this client-supplied structure.
Predicate	Specifies the procedure that is called to determine if the next event in the queue matches the one event specified in the *Argument* parameter.

Return Values

False	Indicates that no event on the queue matched the specified event.
True	Indicates that an event on the queue matched the specified event.

Error Code

BadImplementation

Implementation Specifics

This Xlib subroutine is part of AIXwindows Run Time Environment in AIXwindows Environment/6000.

XInitExtension Subroutine

Purpose

Determines if the extension exists.

Library

Enhanced X-Windows Library (**libX11.a**)

Syntax

XExtCodes *XInitExtension (*DisplayPtr*, *Name*);
Display **DisplayPtr*;
char **Name*;

Description

The **XInitExtension** subroutine determines if the extension exists. It then allocates storage for maintaining the information about the extension on the connection, chains this onto the extension list for the connection, and returns the information needed to access the extension. If the extension does not exist, the **XInitExtension** subroutine returns **NULL**.

Parameters

DisplayPtr Specifies the display.

Name Specifies the name of the extension.

Implementation Specifics

This Xlib subroutine is part of AIXwindows Run Time Environment in AIXwindows Environment/6000.

Related Information

The **_XExtCodes** data structure.

XInsertModifiermapEntry Subroutine

Purpose

Adds an entry to the **XModifierKeymap** structure.

Libraries

Enhanced X-Windows Library (**libX11.a**)

FORTRAN 77 Library (**libXfx.a**)

C Syntax

XModifierKeymap *XInsertModifiermapEntry(*Modifiermap***,** *Keycode***,** *Modifier***)**
XModifierKeymap **Modifiermap*;*
KeyCode *Keycode*;
int *Modifier*;

FORTRAN Syntax

integer*4 fxinsertmodifiermapentry
external fxinsertmodifiermapentry
integer*4 *Modifiermap*
integer*4 *Keycode*
integer*4 *Modifier*
integer*4 *Keymap*
Keymap = **fxinsertmodifiermapentry(***Modifermap***,** *Keycode***,** *Modifier***)**

Description

The **XInsertModifiermapEntry** subroutine adds the specified keycode to the set that controls the specified modifier. This subroutine returns the result to the **XModifierKeymap** structure (expanded as needed).

Parameters

Modifiermap	Specifies a pointer to the **XModifierKeymap** structure.
Keycode	Specifies the keycode.
Modifier	Specifies the modifier.

Error Code

BadImplementation

Implementation Specifics

This Xlib subroutine is part of AIXwindows Run Time Environment in AIXwindows Environment/6000.

Related Information

The **XModifierKeymap** data structure.

XInstallColormap Subroutine

Purpose

Installs a colormap.

Libraries

Enhanced X-Windows Library (**libX11.a**)

FORTRAN 77 Library (**libXfx.a**)

C Syntax

XInstallColormap(*DisplayPtr*, *ColormapID*)
DisplayPtr **DisplayPtr*;
Colormap *ColormapID*;

FORTRAN Syntax

external fxinstallcolormap
integer*4 *DisplayPtr*
integer*4 *ColormapID*
call fxinstallcolormap(*DisplayPtr*, *ColormapID*)

Description

The **XInstallColormap** subroutine installs the specified colormap for its associated screen. All windows associated with this colormap immediately display with true colors. Colormaps become associated with windows through the **XCreateWindow**, **XCreateSimpleWindow**, **XChangeWindowAttributes**, or **XSetWindowColormap** subroutine.

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 each window that has the *ColormapID* parameter as its resource ID. In addition, for every other colormap that is installed as a result of a call to the **XInstallColormap** subroutine, the X Server generates a **ColormapNotify** event on each window that has the uninstalled colormap as its resource ID.

Parameters

ColormapID Specifies the colormap ID.

DisplayPtr Specifies the connection to the X Server.

Error Codes

BadColor

BadImplementation

Implementation Specifics

This Xlib subroutine is part of AIXwindows Run Time Environment in AIXwindows Environment/6000.

Related Information

The **XCreateWindow** subroutine, **XCreateSimpleWindow** subroutine, **XChangeWindowAttributes** subroutine, **XSetWindowColormap** subroutine.

The **InstallColormap** protocol request.

XInternAtom Subroutine

Purpose

Gets an atom for the specified name.

Libraries

Enhanced X-Windows Library (**libX11.a**)

FORTRAN 77 Library (**libXfx.a**)

C Syntax

Atom XInternAtom(*DisplayPtr*, *AtomName*, *OnlyIfExists***)**
Display ******DisplayPtr*;
char ******AtomName*;
Bool *OnlyIfExists*;

FORTRAN Syntax

integer*4 fxinternatom
external fxinternatom
integer*4 *DisplayPtr*
character*256 *AtomName*
integer*4 *OnlyIfExists*
integer*4 *Id*
Id **= fxinternatom(***DisplayPtr*, *AtomName*, *OnlyIfExists***)**

Description

The **XInternAtom** subroutine returns an atom identifier associated with the specified *AtomName* parameter string. You should use a null-terminated ISO Latin-1 string for the *AtomName* parameter. The *AtomName* parameter is case-sensitive. For example the strings "*thing*", "*Thing*" and "*thinG*" all designate different atoms. The atom will remain defined even after the client's connection closes. It will become undefined only when the last connection to the X Server closes.

Parameters

DisplayPtr — 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 if the **XInternAtom** subroutine creates the atom. The *OnlyIfExists* parameter can be as follows:

True — The **XInternAtom** subroutine returns the atom specified in the *AtomName* parameter.

False — The **XInternAtom** subroutine creates the atom if it does not exist.

Return Values

None — The specified atom does not exist.

Atom	The value of the atom.

Error Codes

BadAlloc

BadImplementation

BadValue

Implementation Specifics

This Xlib subroutine is part of AIXwindows Run Time Environment in AIXwindows Environment/6000.

Related Information

The **InternAtom** protocol request.

XIntersectRegion Subroutine

Purpose

Computes the intersection of two regions.

Libraries

Enhanced X-Windows Library (**libX11.a**)

FORTRAN 77 Library (**libXfx.a**)

C Syntax

XIntersectRegion(*SourceA*, *SourceB*, *DestinationReturn*)
Region *SourceA*, *SourceB*, *DestinationReturn*;

FORTRAN Syntax

external fxintersectregion
integer*4 *SourceA, SourceB, DestinationRegion*
call fxintersectregion(*SourceA*, *SourceB*, *DestinationRegion*)

Description

The **XIntersectRegion** subroutine computes the intersection of two regions.

Parameters

DestinationReturn	Stores the result of the computation.
SourceA	Specifies one of the two regions with which to perform the computation.
SourceB	Specifies one of the two regions with which to perform the computation.

Error Code

BadImplementation

Implementation Specifics

This Xlib subroutine is part of AIXwindows Run Time Environment in AIXwindows Environment/6000.

XKeycodeToKeysym Subroutine

Purpose

Converts the KeyCode to a KeySym value.

Library

Enhanced X-Windows Library (**libX11.a**)

FORTRAN 77 Library (**libXfx.a**)

C Syntax

KeySym XKeycodeToKeysym(*DisplayPtr*, *Keycode*, *Index***)**
Display ****DisplayPtr*;**
KeyCode *Keycode*;
int *Index*;

FORTRAN Syntax

integer*4 fxkeycodetokeysym
external fxkeycodetokeysym
integer*4 *DisplayPtr*
integer*4 *Keycode, IndexReturn*
integer*4 *Keysym*
Keysym = **fxkeycodetokeysym(***DisplayPtr*, *Keycode*, *IndexReturn***)**

Description

The **XKeycodeToKeysym** subroutine converts a key code received in a keyboard event to the key symbol that is engraved on the physical key. It uses the internal **Xlib** tables received from the X Server. The keysyms are defined in the **</usr/include/X11/keysymdef.h>** header file.

Parameters

DisplayPtr Specifies the connection to the X Server.

Keycode Specifies the key code.

Index Specifies the column of the **Xlib** library's two-dimensional keycode-to-keysym array. The *Index* parameter can be one of the following values:

0 Specifies the base state (no modifier keys are applied).

1 Specifies the shift state (the shift modifier key is applied).

2 Specifies the mode_switch base state (the mode_switch modifier is applied).

3 Specifies the mode_switch shift state (the mode_switch, and the shift modifiers are applied).

Return Values

NoSymbol	No symbol is defined.
KeySym	The key symbol corresponding to *Keycode*.

Error Code

BadImplementation

Implementation Specifics

This Xlib subroutine is part of AIXwindows Run Time Environment in AIXwindows Environment/6000.

Related Information

The **IsCursorKey** macro.

The **XKeysymToKeycode** subroutine, **XKeysymToString** subroutine, **XLookupKeysym** subroutine, **XLookupString** subroutine, **XStringToKeysym** subroutine, **XRebindKeysym** subroutine.

XKeysymToKeycode Subroutine

Purpose

Converts KeySym value to KeyCode.

Libraries

Enhanced X-Windows Library (**libX11.a**)

FORTRAN 77 Library (**libXfx.a**)

Syntax

KeyCode XKeysymToKeycode(*DisplayPtr*, *Keysym*)
**Display ****DisplayPtr*;
Keysym *Keysym*;

Syntax

integer*4 fxkeysymtokeycode
external fxkeysymtokeycode
integer*4 *DisplayPtr*
integer*4 *Keysym*
integer*4 *Keycode*
Keycode = **fxkeysymtokeycode(***DisplayPtr*, *Keysym*)

Description

The **XKeysymToKeycode** subroutine converts a key symbol value to the appropriate key code and returns the key code.

Parameters

DisplayPtr	Specifies the connection to the X Server.
Keysym	Specifies the key symbol that is to be searched for.

Return Values

0	The key symbol value specified is not defined for any key code.
Keycode	The key code corresponding to the specified *Keysymbol*.

Error Code

BadImplementation

Implementation Specifics

This Xlib subroutine is part of AIXwindows Run Time Environment in AIXwindows Environment/6000.

XKeysymToString Subroutine

Purpose

Converts the KeySym value to the KeySym name.

Libraries

Enhanced X-Windows Library (**libX11.a**)

FORTRAN 77 Library (**libXfx.a**)

C Syntax

char *XKeysymToString(*Keysym***)**
KeySym *Keysym*;

FORTRAN Syntax

character*256 fxkeysymtostring
external fxkeysymtostring
integer*4 *Keysym*
character*256 *KeysymName*
KeysymName = **fxkeysymtostring(***Keysym***)**

Description

The **XKeysymToString** subroutine converts a key symbol code to the name of the key symbol. The name string returned is in a static area and must not be modified.

Parameter

Keysym	Specifies the key symbol to be converted.

Return Values

NULL	The specified key symbol is not defined.
String	The string corresponding to the specified the *KeySym* parameter.

Error Code

BadImplementation

Implementation Specifics

This Xlib subroutine is part of AIXwindows Run Time Environment in AIXwindows Environment/6000.

XKillClient Subroutine

Purpose

Forces a closedown of a client.

Libraries

Enhanced X-Windows Library (**libX11.a**)

FORTRAN 77 Library (**libXfx.a**)

C Syntax

XKillClient(*DisplayPtr*, *Resource*)
Display **DisplayPtr*;
XID *Resource*;

FORTRAN Syntax

external fxkillclient
integer*4 *DisplayPtr*
integer*4 *Resource*
call fxkillclient(*DisplayPtr*, *Resource*)

Description

The **XKillClient** subroutine forces a closedown of the client that created the resource if a valid resource is specified.

If the client has terminated already in the **RetainPermanent** or **RetainTemporary** value mode, all of the client resources are destroyed.

If the **AllTemporary** value is specified, the resources of all clients that have terminated in the **RetainTemporary** value mode are destroyed and the server is reset. This permits implementation of window manager facilities that aid debugging. A client can set its closedown mode to the **RetainTemporary** value. If the client then crashes, its windows would not be destroyed. The programmer can then inspect the application window tree and use the window manager to destroy the zombie windows.

Parameters

DisplayPtr Specifies the connection to the X Server.

Resource Specifies any resource associated with the client to be destroyed.

Error Codes

BadImplementation

BadValue

Implementation Specifics

This Xlib subroutine is part of AIXwindows Run Time Environment in AIXwindows Environment/6000.

Related Information

The **KillClient** protocol request.

XListFonts Subroutine

Purpose

Gets a list of available font names.

Libraries

Enhanced X-Windows Library (**libX11.a**)

FORTRAN 77 Library (**libXfx.a**)

C Syntax

char **XListFonts (*DisplayPtr*, *Pattern*, *MaximumNames*, *ActualCountReturn*)
Display **DisplayPtr*;
char **Pattern*;
int *MaximumNames*;
int **ActualCountReturn*;

FORTRAN Syntax

integer*4 fxlistfonts
external fxlistfonts
integer*4 *DisplayPtr*
character*256 *Pattern*
integer*4 *MaximumNames*, *ActualCountReturn*
integer*4 *FontInformation*
FontInformation = **fxlistfonts**(*DisplayPtr*, *Pattern*, *MaximumNames*, *ActualCountReturn*)

Description

The **XListFonts** subroutine returns an array of font names (as controlled by the font search path; see the **XSetFontPath** subroutine) that matches the string passed to the *Pattern* parameter. The string should be ISO Latin-1 character set; uppercase and lowercase do not matter. Each string is terminated by the ASCII **NULL** value. The pattern string can contain any characters, but each * (asterisk) is a pattern-matching character for any number of characters, and each ? (question mark) is a pattern-matching character for a single character. The client should call the **XFreeFontNames** subroutine when this subroutine is completed.

Parameters

DisplayPtr	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 pattern-matching character on any number of characters, or a ? (question mark), which indicates a pattern-matching character on a single character.
MaximumNames	Specifies the maximum number of names to be in the returned list.
ActualCountReturn	Returns the actual number of font names.

Return Value

List of font names as specified in *ActualCountReturn* long.

Error Code

BadImplementation

Implementation Specifics

This Xlib subroutine is part of AIXwindows Run Time Environment in AIXwindows Environment/6000.

Related Information

The **XChar2b** data structure.

The **XFreeFontNames** subroutine.

The **ListFonts** protocol request.

XListFontsWithInfo Subroutine

Purpose

Gets names and information about fonts.

Libraries

Enhanced X-Windows Library (**libX11.a**)

FORTRAN 77 Library (**libXfx.a**)

C Syntax

char **XListFontsWithInfo(*DisplayPtr*, *Pattern*, *MaximumNames*, *CountReturn*, *InformationReturn*;
**Display ****DisplayPtr*;
**char ****Pattern*;
int *MaximumNames*;
**int ****CountReturn*;
**XFontStruct *****InformationReturn*;

FORTRAN Syntax

integer*4 fxlistfontswithinfo
external fxlistfontswithinfo
integer*4 *DisplayPtr*
character*256 *Pattern*
integer*4, *MaximumNames*, *CountReturn*, *InformationReturn*
integer*4 *FontInformation*
FontInformation = **fxlistfontswithinfo(***DisplayPtr*, *Pattern*, *MaximumNames*, *CountReturn*, *InformationReturn*)

Description

The **XListFontsWithInfo** subroutine returns a list of names of fonts that match the specified pattern and their associated font information. The list of names is limited to the size specified in the *MaximumNames* parameter. The information returned for each font is identical to what the **XLoadQueryFont** subroutine would return except that the per-character metrics are not returned. The pattern string can contain any characters, but each * (asterisk), indicates a pattern-matching character on any number of characters, and each ? (question mark), indicates a pattern-matching character on a single character. Note that the only matching occurs with the pattern string not with any font information.

To free the allocated name array, the client should call the **XFreeFontNames** subroutine. To free the font information array, the client should call the **XFreeFontInfo** subroutine.

Parameters

DisplayPtr	Specifies the connection to the X Server.
Pattern	Specifies the null-terminated pattern string that can contain pattern matching characters.
MaximumNames	Specifies the maximum number of names to be in the returned list.
CountReturn	Returns the actual number of matched font names.

InformationReturn Returns a pointer to the font information.

Return Value

List of font names as specified in *CountReturn* long.

Error Code

BadImplementation

Implementation Specifics

This Xlib subroutine is part of AIXwindows Run Time Environment in AIXwindows Environment/6000.

Related Information

The **XChar2b** data structure, **XFontStruct** data structure.

The **XLoadQueryFont** subroutine, **XFreeFontNames** subroutine, **XFreeFontInfo**subroutine.

The **ListFontsWithInfo** protocol request.

XListHosts Subroutine

Purpose

Gets the list of hosts.

Libraries

Enhanced X-Windows Library (**libX11.a**)

FORTRAN 77 Library (**libXfx.a**)

C Syntax

XHostAddress *XListHosts(*DisplayPtr*, *NumberHostsReturn*, *StateReturn***)**
**Display ***DisplayPtr*;
**int ***NumberHostsReturn*;
**Bool ***StateReturn*;

FORTRAN Syntax

integer*4 fxlisthosts
external fxlisthosts
integer*4 *DisplayPtr*, *NumberHostsReturn*, *StateReturn*
integer*4 *HostAddress*
HostAddress **= fxlisthosts(***DisplayPtr*, *NumberHostsReturn*, *StateReturn***)**

Description

The **XListHosts** subroutine returns the current access control list and the state of the control which indicates if connection setup was enabled or disabled. This subroutine 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.

Use the **XFree** subroutine to free the allocated memory when it is no longer needed.

Parameters

DisplayPtr	Specifies the connection to the X Server.
NumberHostsReturn	Returns the number of hosts currently in the access control list.
StateReturn	Returns the state (enabled or disabled) of the access control list.

Error Code

BadImplementation

Implementation Specifics

This Xlib subroutine is part of AIXwindows Run Time Environment in AIXwindows Environment/6000.

Related Information

The **XHostAddress** data structure.

The **XFree** subroutine.

The **ListHosts** protocol request.

XListInstalledColormaps Subroutine

Purpose

Gets a list of currently installed colormaps for a given screen.

Libraries

Enhanced X-Windows Library (**libX11.a**)

FORTRAN 77 Library (**libXfx.a**)

C Syntax

Colormap *XListInstalledColormaps(*DisplayPtr*, *WindowID*, *NumberReturn***)**
**Display ***DisplayPtr*;
Window *WindowID*;
**int ***NumberReturn*;

FORTRAN Syntax

integer*4 fxlistinstalledcolormaps
external fxlistinstalledcolormaps
integer*4 *DisplayPtr*
integer*4 *WindowID*, *NumberReturn*
integer*4 *Colormap*
Colormap = **fxlistinstalledcolormaps(***DisplayPtr*, *WindowID*, *NumberReturn***)**

Description

The **XListInstalledColormaps** subroutine 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 the **XFree** subroutine to free the allocated list when it is no longer needed.

Parameters

DisplayPtr	Specifies the connection to the X Server.
WindowID	Specifies the window ID for the window for which a screen list of currently installed colormaps is to be obtained.
NumberReturn	Returns the list of currently installed colormaps.

Return Value

A list of installed colormaps as specified in the *NumberReturn* parameter.

Error Codes

BadImplementation

BadWindow

Implementation Specifics

This Xlib subroutine is part of AIXwindows Run Time Environment in AIXwindows Environment/6000.

Related Information

The **XFree** subroutine.

The **ListInstalledColormaps** protocol request.

XListProperties Subroutine

Purpose

Gets the specified window property list.

Libraries

Enhanced X-Windows Library (**libX11.a**)

FORTRAN 77 Library **(libXfx.a**)

C Syntax

Atom *XListProperties(*DisplayPtr, WindowID, NumberPropertiesReturn***)**
**Display ***DisplayPtr;
Window WindowID;
**int ***NumberPropertiesReturn*;

FORTRAN Syntax

integer*4 fxlistproperties
external fxlistproperties
integer*4 *DisplayPtr*
integer*4 *WindowID*
integer*4 *NumberPropertiesReturn*
Atom = **fxlistproperties(***DisplayPtr, WindowID, NumberPropertiesReturn***)**

Description

The **XListProperties** subroutine 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.

Use the **XFree** subroutine to free the allocated memory when it is no longer needed.

Parameters

DisplayPtr	Specifies the connection to the X Server.
NumberPropertiesReturn	Returns the length of the properties list.
WindowID	Specifies the window ID for the window for which a property list is to be obtained.

Return Values

NULL	No properties were found.

List of the atoms of properties on the specified window as indicated in the *NumberPropertiesReturn* parameter.

Error Codes

BadImplementation

BadWindow

Implementation Specifics

This Xlib subroutine is part of AIXwindows Run Time Environment in AIXwindows Environment/6000.

Related Information

The **XFree** subroutine.

The **ListProperties** protocol request.

XLoadFont Subroutine

Purpose

Loads a font.

Libraries

Enhanced X-Windows Library (**libX11.a**)

FORTRAN 77 Library (**libXfx.a**)

C Syntax

Font XLoadFont(*DisplayPtr*, *Name***)**
**Display ****DisplayPtr*;
**char ****Name*;

FORTRAN Syntax

integer*4 fxloadfont
external fxloadfont
integer*4 *DisplayPtr*
character*256 *Name*
integer*4 *Font*
Font **= fxloadfont(***DisplayPtr*, *Name***)**

Description

The **XLoadFont** subroutine loads the specified font and returns the associated font ID. The name should be ISO Latin-1 encoding; it is not case-sensitive. Use the **XUnloadFont** subroutine 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.

Parameters

DisplayPtr	Specifies the connection to the X Server.
Name	Specifies the font name (a null-terminated string).

Error Codes

BadAlloc

BadImplementation

BadName

Implementation Specifics

This Xlib subroutine is part of AIXwindows Run Time Environment in AIXwindows Environment/6000.

Related Information

The **XChar2b** data structure.

The **XUnloadFont** subroutine.

The **OpenFont** protocol request.

XLoadQueryFont Subroutine

Purpose

Loads and queries a font in one operation.

Libraries

Enhanced X–Windows Library (**libX11.a**)

FORTRAN 77 Library (**libXfx.a**)

C Syntax

XFontStruct *XLoadQueryFont(*DisplayPtr*, *Name***)**
**Display ***DisplayPtr*;
**char ***Name*;

FORTRAN Syntax

integer*4 fxloadqueryfont
external fxloadqueryfont
integer*4 *DisplayPtr*
character*256 *Name*
integer*4 *XFontStruct*
XFontStruct **= fxloadqueryfont(***DisplayPtr*, *Name***)**

Description

The **XLoadQueryFont** subroutine 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.

The **XFontStruct** structure contains all the font information and a pointer to an array of **XCharStruct** structures for the characters contained in the font.

Parameters

DisplayPtr	Specifies the connection to the X Server.
Name	Specifies the font name (a null–terminated string).

Return Values

NULL The specified font does not exist.

Pointer to the **XFontStruct** structure.

Error Codes

BadAlloc

BadImplementation

Implementation Specifics

This Xlib subroutine is part of AIXwindows Run Time Environment in AIXwindows Environment/6000.

Related Information

The **XChar2b** data structure, **XFontStruct** data structure.

The **XFreeFont** subroutine.

The **OpenFont**protocol request, **QueryFont** protocol request.

XLookUpAssoc Subroutine

Purpose

Obtains data from a specific associate table.

Library

Enhanced X-Windows Library (**liboldX.a**)

Syntax

```
#include <X11/X10.h>
char *XLookUpAssoc(DisplayPtr, Table, x_id)
 Display *DisplayPtr;
 XAssocTable *Table;
 XID x_id;
```

Description

The **XLookUpAssoc** subroutine obtains data from a specific associate table. It retrieves the data stored in an **XAssocTable** structure by its XID. If an appropriately matching XID is found in the table the subroutine returns the data associated with it. If the XID cannot be found in the table the subroutine returns NULL.

Note: This subroutine is in the **liboldX.a** library. Include this library in the compiler command to build your program. For example:

```
{compiler-option} -o samples samples.c -loldX -lX11
```

Parameters

DisplayPtr	Specifies the connection to the X Server.
Table	Specifies the associate table.
x_id	Specifies the XID.

Implementation Specifics

This Xlib subroutine is part of AIXwindows Run Time Environment in AIXwindows Environment/6000.

Related Information

The **XCreateAssocTable** subroutine.

XLookupColor Subroutine

Purpose

Looks up a color name.

Libraries

Enhanced X-Windows Library (**libX11.a**)

FORTRAN 77 Library (**libXfx.a**)

C Syntax

Status XLookupColor(*DisplayPtr*, *ColorMapID*, *ColorName*, *ExactDefinitionReturn*, *ScreenDefinitionReturn*)
Display **DisplayPtr*;
Colormap *ColorMapID*;
char **ColorName*;
XColor **ExactDefintionReturn*, **ScreenDefinitionReturn*;

FORTRAN Syntax

integer*4 fxlookupcolor
external fxlookupcolor
integer*4 *DisplayPtr*
integer*4 *ColormapID*
character*256 *ColorName*
integer*4 *ScreenDefinedReturn*
integer*4 *ExactDefinedReturn*
integer*4 *Status*
Status = **fxlookupcolor**(*DisplayPtr*, *ColormapID*, *ColorName*, *ScreenDefinedReturn*, *ExactDefinedReturn*)

Description

The **XLookupColor** subroutine looks up the string of a name for the screen associated with the specified *ColorMapID* parameter. It returns the color values and the closest values provided by the screen with respect to the visual type of the specified colormap. You should use the ISO Latin-1 encoding; it is not case-sensitive.

Parameters

ColorMapID	Specifies the colormap ID.
ColorName	Specifies the string of the color name for the color definition structure to be returned.
DisplayPtr	Specifies the connection to the X Server.
ExactDefinitionReturn	Returns the exact RGB values for the color specified in the *ColorName* parameter.
ScreenDefinitionReturn	Returns the closest RGB values provided by the hardware.

Return Values

False	The color is not in the RGB database.
True	The color is in the RGB database.

Error Codes

BadColor

BadImplementation

BadName

Implementation Specifics

This Xlib subroutine is part of AIXwindows Run Time Environment in AIXwindows Environment/6000.

Related Information

The **XColor** data structure.

The **LookupColor** protocol request.

XLookupKeysym Subroutine

Purpose

Translates keyboard event into a key symbol value.

Libraries

Enhanced X-Windows Library (**libX11.a**)

FORTRAN 77 Library (**libXfx.a**)

C Syntax

KeySym XLookupKeysym(*EventKey*, *Index*)
XKeyEvent **EventKey*;
int *Index*;

FORTRAN Syntax

integer*4 fxlookupkeysym
external fxlookupkeysym
integer*4 *EventKey*
integer*4 *Index*
integer*4 *Keysym*
Keysym = **fxlookupkeysym**(*EventKey*, *Index*)

Description

The **XLookupKeysym** subroutine looks up the key symbol. It uses a given keyboard event and the index specified to return the key symbol from the list that corresponds to the *keycode* field in the **XKeyPressedEvent** or the **XKeyReleasedEvent** structure.

Parameters

EventKey	Specifies the key event, which can be either the **KeyPress** or the **KeyRelease** key event to be used.
Index	Specifies the index into the key symbol table.

Return Values

NoSymbol	If no key symbol is defined for the key code of the event.
KeySym	Key symbol corresponding to event.

Error Code

BadImplementation

Implementation Specifics

This Xlib subroutine is part of AIXwindows Run Time Environment in AIXwindows Environment/6000.

XLookupMapping Subroutine

Purpose

Gets mapping of the keyboard event from a keymap file.

Library

Enhanced X-Windows Library (**liboldX.a**)

FORTRAN 77 Library (**libXfx.a**)

C Syntax

char* XLookupMapping (*Event*, *NumberBytes*)
XKeyPressedEvent **Event*;
int **NumberBytes*;

FORTRAN Syntax

external fxlookupmapping
integer*4 fxlookupmapping
integer*4 *Event*
integer*4 *Numberbytes*
integer*4 *Map*
map = **fxlookupmapping** (*Event, NumberBytes*)

Description

The **XLookupMapping** subroutine maps events to counted character strings (an array of characters and the length; the null character is legitimate in this use). The subroutine 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.

The **XLookupMapping** subroutine searches for the current keyboard mapping in the following order of files:

- **$XDIR/imkeymap**
- **$HOME/imkeymap**
- **/usr/lib/nls/im/$LANG/imkeymap**

If these files are not present, the **XLookupMapping** subroutine defaults to the **XLookupString** subroutine.

The **imkeymap** file is produced by the **keycomp** command, which reads a text file of keyboard mappings. The keyboard mappings in this file are based on KeySym values, not Keycode values. Therefore, the first key symbol in the list of KeySyms associated with the key code in the **XKeyPressedEvent** subroutine is used to access the **imkeymap** file. The **/usr/lib/nls/im/$LANG** directory, where **$LANG** is the version 3 environment variable for language/locale, contains the keyboard mappings for languages selected during installation of Enhanced X-Windows.

The **XLookupMapping** subroutine performs normal interpretation of shift bits (alt, shift, shift lock, and control). It supports Alt-NumPad and NumLock key processing as well as the dead key processing defined in the **keycomp** subroutine.

The Alt-NumPad subroutine 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, the **XLookupMapping** subroutine tracks the NumLock state only if the NumLock key is defined as UNBOUND.

Use the **strncpy** command to copy the result for storage if the data must be modified. If a different keymap file is desired, use the **XUseKeymap** subroutine. On the RISC System/6000, the **XLookupMapping** subroutine invokes the Input Method. Thus it is necessary to include the Input Method library (**/usr/lib/libIM.a**) when linking a program by using the flag **–lIM**, as in th efollowing example:

```
cc file1.c file2.c –lX11 –loldX –lIM
```

Parameters

Event Specifies the **KeyPress** event to be used.

NumberBytes Returns a pointer to the number of bytes returned in the character string or a value of 0 if no text is mapped to the event.

Error Code

BadImplementation

Implementation Specifics

This Xlib subroutine is part of AIXwindows Run Time Environment in AIXwindows Environment/6000.

Related Information

The **XLookupString** subroutine, **XUseKeymap** subroutine.

The **keycomp** command.

XLookupString Subroutine

Purpose

Translates a keyboard event into a character string.

Libraries

Enhanced X-Windows Library (**libX11.a**)

FORTRAN 77 Library (**libXfx.a**)

C Syntax

int XLookupString(*EventStructure*, *BufferReturn*, *BytesBuffer*, *KeysymReturn*, *StatusReturn*)

XKeyEvent **EventStructure*;
char **BufferReturn*;
int *BytesBuffer*;
KeySym **KeysymReturn*;
XComposeStatus **StatusReturn*;

FORTRAN Syntax

integer*4 fxlookupstring
external fxlookupstring
integer*4 *EventStructure*
character*256 *BufferReturn*
integer*4 *BytesBuffer*
integer*4 *KeysymReturn*, *StatusReturn*
integer*4 *Stringlength*
StringLength = **fxlookupstring**(*EventStructure*, *BufferReturn*, *BytesBuffer*, *KeysymReturn*, *StatusReturn*)

Description

The **XLookupString** subroutine maps a key event to an ISO Latin-1 string, using the modifier bits in the key event to deal with the Shift, Lock, and Control keys. This subroutine returns the translated string into the user's buffer. The **XLookupString** subroutine also detects any rebound KeySyms 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, the **XLookupString** subroutine interprets the lock modifier to perform Caps Lock processing.

If present, (non-**NULL**) the **XCompose** data structure records the state, which is private to the **Xlib** library, that needs preservation across calls to the **XLookupString** subroutine to implement compose processing.

Parameters

EventStructure	Specifies the key event structure (**XKeyPressedEvent** or **XKeyReleasedEvent**) to be used.
BufferReturn	Returns the translated characters.
BytesBuffer	Specifies the length of the *BufferReturn* parameter.
KeysymReturn	Returns the KeySym computed from the event.
StatusReturn	Specifies the status of the processing. This parameter returns a pointer to the **XCompose** data structure or the value of **NULL.**

Error Code

BadImplementation

Implementation Specifics

This Xlib subroutine is part of AIXwindows Run Time Environment in AIXwindows Environment/6000.

Related Information

XRebindKeysym subroutine.

XLowerWindow Subroutine

Purpose

Lowers the specified window.

Libraries

Enhanced X-Windows Library (**libX11.a**)

FORTRAN 77 Library (**libXfx.a**)

C Syntax

XLowerWindow(*DisplayPtr*, *WindowID*)
Display **DisplayPtr*;
Window *WindowID*;

FORTRAN Syntax

external fxlowerwindow
integer*4 *DisplayPtr*
integer*4 *WindowID*
call fxlowerwindow(*DisplayPtr*, *WindowID*)

Description

The **XLowerWindow** subroutine 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 coordinates location on the desk constant. Lowering a mapped window generates **Expose** events on any formerly obscured windows.

The X Server generates a **ConfigureRequest** event and no processing is performed if the *OverrideRedirect* attribute of the window is a **False** value and another client selected the **SubstructureRedirectMask** event mask on the parent window. Otherwise, the window is lowered to the bottom of the stack.

Parameters

DisplayPtr	Specifies the connection to the X Server.
WindowID	Specifies the window ID of the window to be lowered.

Error Codes

BadImplementation

BadWindow

Implementation Specifics

This Xlib subroutine is part of AIXwindows Run Time Environment in AIXwindows Environment/6000.

Related Information

The **ConfigureWindow** protocol request.

XMakeAssoc Subroutine

Purpose

Creates an entry in a specific associate table.

Library

Enhanced X-Windows Library (**liboldX.a**)

Syntax

```
#include <X11/X10.h>
XMakeAssoc(DisplayPtr, Table, x_id, Data)
 Display *DisplayPtr;
 XAssocTable *Table;
 XID x_id;
 char *Data;
```

Description

The **XDeleteAssoc** subroutine creates an entry in a specific associate table. It inserts data into an **XAssocTable** structure keyed on an XID. Data is inserted 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.

Note: This subroutine is in the **liboldX.a** library. Include this library in the compiler command to build your program. For example:

```
{compiler-option} -o samples samples.c -loldX -lX11
```

Parameters

DisplayPtr	Specifies the connection to the X Server.
Table	Specifies the associate table.
x_id	Specifies the XID.
Data	Specifies the data to be associated with the *x_id* parameter.

Implementation Specifics

This Xlib subroutine is part of AIXwindows Run Time Environment in AIXwindows Environment/6000.

Related Information

The **XCreateAssocTable** subroutine.

XMapRaised Subroutine

Purpose

Maps and raises a specified window.

Libraries

Enhanced X-Windows Library (**libX11.a**)

FORTRAN 77 Library (**libXfx.a**)

C Syntax

XMapRaised(*DisplayPtr*, *WindowID*)
Display **DisplayPtr*;
Window *WindowID*;

FORTRAN Syntax

external fxmapraised
integer*4 *DisplayPtr*
integer*4 *WindowID*
call fxmapraised(*DisplayPtr*, *WindowID*)

Description

The **XMapRaised** subroutine maps a specified window and all of its subwindows that have had map requests. It also raises the specified window to the top of the stack.

Parameters

DisplayPtr Specifies the connection to the X Server.

WindowID Specifies a window ID.

Error Codes

BadImplementation

BadWindow

Implementation Specifics

This Xlib subroutine is part of AIXwindows Run Time Environment in AIXwindows Environment/6000.

Related Information

The **XMapWindow** subroutine.

The **ConfigureWindow** protocol request, **MapWindow** protocol request.

XMapSubwindows Subroutine

Purpose

Maps all subwindows of a specified window.

Libraries

Enhanced X-Windows Library (**libX11.a**)

FORTRAN 77 Library (**libXfx.a**)

C Syntax

XMapSubwindows(*DisplayPtr*, *WindowID***)**
**Display ****DisplayPtr*;
Window *WindowID*;

FORTRAN Syntax

external fxmapsubwindows
integer*4 *DisplayPtr*
integer*4 *WindowID*
call fxmapsubwindows(*DisplayPtr*, *WindowID***)**

Description

The **XMapSubwindows** subroutine maps all subwindows of a specified window in top-to-bottom stacking order.

The X Server generates the **Expose** events on each newly displayed window. Using the **XMapSubwindows** subroutine can be more efficient than individually mapping multiple windows.

Parameters

DisplayPtr Specifies the connection to the X Server.

WindowID Specifies the window ID.

Error Codes

BadImplementation

BadWindow

Implementation Specifics

This Xlib subroutine is part of AIXwindows Run Time Environment in AIXwindows Environment/6000.

Related Information

The **MapSubwindows** protocol request.

XMapWindow Subroutine

Purpose

Maps a specified window.

Libraries

Enhanced X-Windows Library (**libX11.a**)

FORTRAN 77 Library (**libXfx.a**)

C Syntax

XMapWindow(*DisplayPtr*, *WindowID*)
Display **DisplayPtr*;
Window *WindowID*;

FORTRAN Syntax

external fxmapwindow
integer*4 *DisplayPtr*
integer*4 *WindowID*
call fxmapwindow(*DisplayPtr*, *WindowID*)

Description

The **XMapWindow** subroutine maps a specified window and all of its subwindows that have had map requests. Mapping a window that has an unmapped ancestor does not display the window; such a window is unviewable. Mapping marks it as eligible for display once all the ancestors of the window are mapped. The window then becomes viewable and is visible on the screen if it is not obscured by another window. The **XMapWindow** subroutine has no effect on a window that already mapped.

If the *override_redirect* field of a specified window is a value of **False** and if another client has selected the **SubstructureRedirectMask** value on the parent window, the X Server generates a **MapRequest** event and the **XMapWindow** subroutine does not map the window. Otherwise, the X Server generates a **MapNotify** event and the window is mapped.

If the specified window becomes viewable and has no stored contents, the X server tiles the window with its background. If the background is undefined, the existing screen contents are not altered, and the X Server can generate the **Expose** events.

If backing store is maintained while the window is unmapped, no **Expose** events are generated. If a backing store is now maintained, a full window exposure is always generated. Otherwise, only visible regions can be reported. Similar tiling and exposure take place for any newly viewable inferiors.

If the specified window is an **InputOutput** window, the **XMapWindow** subroutine generates the **Expose** events on each **InputOutput** window displayed as a result.

If the client maps and paints the window, and begins processing events, the window is painted twice. To avoid this, request **Expose** events, then map the window, so the client processes input events as usual. The event list includes the **Expose** events for each window displayed on the screen. The normal response of the client to an **Expose** event is to repaint the window.

Parameters

DisplayPtr Specifies the connection to the X Server.

WindowID Specifies the window ID.

Error Codes

BadImplementation

BadWindow

Implementation Specifics

This Xlib subroutine is part of AIXwindows Run Time Environment in AIXwindows Environment/6000.

Related Information

The **MapWindow** protocol request.

XMaskEvent Subroutine

Purpose

Removes the next event that matches a specified event mask.

Libraries

Enhanced X-Windows Library (**libX11.a**)

FORTRAN 77 Library (**libXfx.a**)

C Syntax

XMaskEvent(*DisplayPtr*, *EventMask*, *EventReturn*)
Display **DisplayPtr*;
unsigned long *EventMask*;
XEvent **EventReturn*;

FORTRAN Syntax

external fxmaskevent
integer*4 *DisplayPtr*
integer*4 *EventMask*, *EventReturn*
call fxmaskevent(*DisplayPtr*, *EventMask*, *EventReturn*)

Description

The **XMaskEvent** subroutine searches the event queue for events associated with a specified mask. When it finds a match, it removes matched events, then copies them into the specified **XEvent** data structure. Other events stored in the queue are not discarded.

If a requested event is not in the queue, the **XMaskEvent** subroutine flushes the output buffer, then blocks until one is received.

Parameters

DisplayPtr	Specifies the connection to the X Server.
EventMask	Specifies the event mask.
EventReturn	Returns the associated data structure of the matched event.

Error Code

BadImplementation

Implementation Specifics

This Xlib subroutine is part of AIXwindows Run Time Environment in AIXwindows Environment/6000.

XMatchVisualInfo Subroutine

Purpose

Gets the visual information that matches the specified depth and class of the screen.

Libraries

Enhanced X-Windows Library (**libX11.a**)

FORTRAN 77 Library (**libXfx.a**)

C Syntax

Status XMatchVisualInfo(*DisplayPtr*, *Screen*, *Depth*, *Class*, *VisualInformationReturn***)**
Display **DisplayPtr*;
int *Screen*;
int *Depth*;
int *Class*;
XVisualInfo **VisualInformationReturn*;

FORTRAN Syntax

integer*4 fxmatchvisualinfo
external fxmatchvisualinfo
integer*4 *DisplayPtr*, *Screen*
integer*4 *Depth*, *Class*
integer*4 *VisualInformationReturn*
integer*4 *Status*
Status **= fxmatchvisualinfo(***DisplayPtr*, *Screen*, *Depth*, *Class*, *VisualInformationReturn***)**

Description

The **XMatchVisualInfo** subroutine obtains the visual information for a visual that matches the specified depth and class of the screen. Since multiple visuals that match the specified depth and class can exist, the exact visual chosen is undefined. If a visual that matches is found, the **XMatchVisualInfo** subroutine returns the information on the visual to the *VisualInformationReturn* parameter.

Parameters

Class	Specifies the class of the screen.
Depth	Specifies the depth of the screen.
DisplayPtr	Specifies the connection to the X Server.
Screen	Specifies the screen.
VisualInformationReturn	Returns the matched visual information.

Return Values

False	A visual that matches is not found.
True	A visual that matches is found.

Error Code

BadImplementation

Implementation Specifics

This Xlib subroutine is part of AIXwindows Run Time Environment in AIXwindows Environment/6000.

XMoveResizeWindow Subroutine

Purpose

Changes the size and location of a specified window.

Libraries

Enhanced X-Windows Library (**libX11.a**)

FORTRAN 77 Library (**libXfx.a**)

C Syntax

XMoveResizeWindow(*DisplayPtr*, *WindowID*,
X, *Y*, *Width*, *Height*)
Display **DisplayPtr*;
Window *WindowID*;
int *X*, *Y*;
unsigned int *Width*, *Height*;

FORTRAN Syntax

external fxmoveresizewindow
integer*4 *DisplayPtr*
integer*4 *WindowID*
integer*4 *X*
integer*4 *Y*
integer*4 *Width*
integer*4 *Height*
call fxmoveresizewindow(*DisplayPtr*, *WindowID*, *X*, *Y*, *Width*, *Height*)

Description

The **XMoveResizeWindow** subroutine changes the size and location of a specified window without raising it. Moving and resizing a mapped window can generate an **Expose** event on the window. Depending on the new size and location parameters, moving and resizing a window can generate exposure events on windows that the window formerly obscured.

If the *override_redirect* field of the window is a value of **False** and another client has selected the **SubstructureRedirectMask** mask on the parent window, the X Server generates a **ConfigureRequest** event; no further processing is performed. Otherwise, the window size and location are changed.

The *X* and *Y* parameters define the new position of the window relative to its parent window. The *Width* and *Height* parameters define the interior size of the window.

Parameters

DisplayPtr	Specifies the connection to the X Server.
Height	Specifies the new height of the window.
Width	Specifies the new width of the window.
WindowID	Specifies the window to be reconfigured.
X	Specifies the x coordinate for the new position of the window relative to the parent.
Y	Specifies the y coordinate for the new position of the window relative to the parent.

Error Codes

BadImplementation

BadValue

BadWindow

Implementation Specifics

This Xlib subroutine is part of AIXwindows Run Time Environment in AIXwindows Environment/6000.

Related Information

The **ConfigureWindow** protocol request.

Configuring Enhanced X-Windows Windows

XMoveWindow Subroutine

Purpose

Moves a specified window without changing its size.

Libraries

Enhanced X-Windows Library (**libX11.a**)

FORTRAN 77 Library (**libXfx.a**)

C Syntax

XMoveWindow(DisplayPtr, *WindowID*, *X*, *Y*)
Display **DisplayPtr*;
Window *WindowID*;
int *X*, *Y*;

FORTRAN Syntax

external fxmovewindow
integer*4 *DisplayPtr*
integer*4 *WindowID*
integer*4 *X*
integer*4 *Y*
call fxmovewindow(*DisplayPtr*, *WindowID*, *X*, *Y*)

Description

The **XMoveWindow** subroutine moves the specified window to the coordinates specified by the *X* and *Y* parameters. It does not change the size or mapping state of the window, and does not raise the window.

A mapped window can lose its contents when moved and one of the following occurs:

- If its *background_pixmap* field is the **parentRelative** value
- If it is obscured by a non-child window and has no backing store.

If the contents of the window are lost, the X Server generates the **Expose** events. Moving a mapped window generates the **Expose** events on any formerly obscured windows.

If the *override_redirect* field of the window is a value of **False** and another client has selected the **SubstructureRedirectMask** on the parent window, the X Server generates a **ConfigureRequest** event, and no further processing is performed. Otherwise, the window is moved.

The *X* and *Y* parameters define the new location for either the top-left pixel of the window border or of the window itself, if it has no border.

Parameters

DisplayPtr	Specifies the connection to the X Server.
WindowID	Specifies the window to be moved.
X	Specifies the x coordinate for the new location of the window.
Y	Specifies the y coordinate for the new location of the window

Error Codes

BadImplementation

BadWindow

Implementation Specifics

This Xlib subroutine is part of AIXwindows Run Time Environment in AIXwindows Environment/6000.

Related Information

The **ConfigureWindow** protocol request.

XNewModifiermap Subroutine

Purpose

Creates an **XModifierKeymap** data structure.

Libraries

Enhanced X-Windows Library (**libX11.a**)

FORTRAN 77 Library (**libXfx.a**)

C Syntax

XModifierKeymap *XNewModifiermap(*MaximumKeysPerModifiers***)**
int *MaximumKeysPerModifiers*;

FORTRAN Syntax

integer*4 fxnewmodifiermapping
external fxnewmodifiermapping
integer*4 *MaximumKeysPerModifier*
integer*4 *Modifiermap*
Modifiermap = **fxnewmodifiermapping(***MaximumKeysPerModifier***)**

Description

The **XNewModifiermap** subroutine returns a pointer to an **XModifierKeymap** data structure for later use.

Use the **XFreeModifierMap** subroutine to free the storage when the modifier map is no longer needed.

Parameter

MaximumKeysPerModifiers	Specifies the maximum number of key codes assigned to any of the modifiers in the map.

Error Code

BadImplementation

Implementation Specifics

This Xlib subroutine is part of AIXwindows Run Time Environment in AIXwindows Environment/6000.

XNextEvent Subroutine

Purpose

Gets the next event and removes it from the queue.

Libraries

Enhanced X-Windows Library (**libX11.a**)

FORTRAN 77 Library (**xlibXfx.a**)

C Syntax

XNextEvent(*DisplayPtr*, *EventReturn*)
Display **DisplayPtr*;
XEvent **EventReturn*;

FORTRAN Syntax

external fxnextevent
integer*4 *DisplayPtr*
integer*4 *EventReturn*
call fxnextevent(*DisplayPtr*, *EventReturn*)

Description

The **XNextEvent** subroutine copies the first event from the event queue into a specified **XEvent** data structure, then removes it from the queue. If the event queue is empty, the **XNextEvent** subroutine flushes the output buffer, then blocks until an event is received.

Parameters

DisplayPtr	Specifies the connection to the X Server.
EventReturn	Returns the next event in the queue.

Error Code

BadImplementation

Implementation Specifics

This Xlib subroutine is part of AIXwindows Run Time Environment in AIXwindows Environment/6000.

XNoOp Subroutine

Purpose

Sends a **NoOperation** protocol request to the X Server.

Libraries

Enhanced X-Windows Library (**libX11.a**)

FORTRAN 77 Library (**libXfx.a**)

C Syntax

XNoOp(*DisplayPtr***)**
Display ***DisplayPtr;

FORTRAN Syntax

external fxnoop
integer*4 *DisplayPtr*
call fxnoop(*DisplayPtr***)**

Description

The **XNoOp** subroutine sends a **NoOperation** protocol request to the X Server in order to check the connection to the display system. It does not flush the output buffer.

Parameter

DisplayPtr Specifies the connection to the X Server.

Implementation Specifics

This Xlib subroutine is part of AIXwindows Run Time Environment in AIXwindows Environment/6000.

Related Information

The **NoOperation** protocol request.

XOffsetRegion Subroutine

Purpose

Moves a region by a specified amount.

Libraries

Enhanced X-Windows Library (**libX11.a**)

FORTRAN 77 Library (**libXfx.a**)

C Syntax

XOffsetRegion(*RegionPtr*, *DefineX*, *DefineY*)
Region *RegionPtr*;
int *DefineX*, *DefineY*;

FORTRAN Syntax

external fxoffsetregion
integer*4 *RegionPtr*, *DefineX*, *DefineY*
call fxoffsetregion(*RegionPtr*, *DefineX*, *DefineY*)

Description

The **XOffsetRegion** subroutine moves a specified region by a specified amount. The coordinates specified in the *DefineX* and *DefineY* parameters define the amount to move the specified region.

Parameters

RegionPtr	Specifies the region.
DefineX	Specifies the x coordinate.
DefineY	Specifies the y coordinate.

Error Code

BadImplementation

Implementation Specifics

This Xlib subroutine is part of AIXwindows Run Time Environment in AIXwindows Environment/6000.

XOpenDisplay Subroutine

Purpose

Opens a connection to the X Server that controls a display device.

Libraries

Enhanced X-Windows Library (**libX11.a**)

FORTRAN 77 Library (**libXfx.a**)

C Syntax

Display *XOpenDisplay(*DisplayName***)**
char ***DisplayName*;**

FORTRAN Syntax

integer*4 fxopendisplay
external fxopendisplay
integer*4 *Display*
character*6 *Hostname*
Hostname **= 'unix:0'**
Display **= fxopendisplay(***Hostname***)**

Description

The **XOpenDisplay** subroutine opens a connection to the X Server controlling the specified display device. The *DisplayName* parameter establishes the display and communications domain to be used. The X Server can implement various types of access control mechanisms that allow clients to use the screens in the display.

On a UNIX based system, the host name is of the following format:

HostName:*Number*.*ScreenNumber*

If the *DisplayName* parameter is the **NULL** value, the **XOpenDisplay** subroutine uses the **DISPLAY** environment variable. If the **DISPLAY** environment variable is the **NULL** value, the **XOpenDisplay** subroutine uses the default display name.

If successful, the **XOpenDisplay** subroutine returns a pointer to a **Display** data structure, defined in the **<X11/Xlib.h>** file. If the **XOpenDisplay** subroutine is not successful, it returns a **NULL** value, and the value of the *DisplayName* parameter defaults to the **DISPLAY** environment variable.

After a successful call to the **XOpenDisplay** subroutine, all the screens in the display can be used by the client. The screen number specified in the *DisplayName* parameter is returned by the **DefaultScreen** macro or the **XDefaultScreen** subroutine. Elements of the **Display** and **Screen** data structures can only be accessed by using the information macros or functions.

Parameter

DisplayName Specifies the display device.

Return Values

NULL	The **XOpenDisplay** subroutine is not successful.
Pointer to display structure	The **XOpenDisplay** subroutine is successful.

Error Code

BadImplementation

Implementation Specifics

This Xlib subroutine is part of AIXwindows Run Time Environment in AIXwindows Environment/6000.

Related Information

The **CreateGC** protocol request.

The **DefaultScreen** macro.

XParseColor Subroutine

Purpose

Creates RGB values from color name strings.

Libraries

Enhanced X-Windows Library (**libX11.a**)

FORTRAN 77 Library (**libXfx.a**)

C Syntax

Status XParseColor(*DisplayPtr*, *ColormapID*, *Specification*, *ExactDefinitionReturn*)
Display **DisplayPtr*;
Colormap *ColormapID*;
char **Specification*;
XColor **ExactDefinitionReturn*;

FORTRAN Syntax

integer*4 fxparsecolor
external fxparsecolor
integer*4 *DisplayPtr*, *ColormapID*
character*256 *Specification*
integer*4 *ExactDefinitionReturn*
integer*4 *Status*
Status = **fxparsecolor**(*DisplayPtr*, *ColormapID*, *Specification*, *ExactDefinitionReturn*)

Description

The **XParseColor** subroutine creates a standard user interface to color. It takes a string specification of a color, typically from a command line or from the *Option* parameter of the **XGetDefault** subroutine, and returns the corresponding RGB (red, green, and blue) values that are suitable for a subsequent call to the **XAllocColor** or **XStoreColor** subroutines.

The color can be specified as a color name, as in the **XAllocNamedColor** subroutine; or, it can be specified by an initial sharp sign character followed by a numeric specification, as in one of the following formats:

`#RGB`	(4 bits each)
`#RRGGBB`	(8 bits each)
`#RRRGGGBBB`	(12 bits each)
`#RRRRGGGGBBBB`	(16 bits each)

In this format, the `R`, `G`, and `B` values represent single hexadecimal digits (upper or lower case). 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 on which to look up the color. The default colormap for the screen can be used.

The *ExactDefinitionReturn* parameter returns the exact color value for later use and sets the **DoRed**, **DoGreen**, and **DoBlue** flags of the **XColor** data structure.

Parameters

ColormapID	Specifies the colormap ID.
DisplayPtr	Specifies the connection to the X Server.
ExactDefinitionReturn	Returns the exact color value.
Specification	Specifies the color name as a string (not case sensitive).

Return Values

True	The **XParseColor** subroutine is successful.
False	The **XParseColor** subroutine is not successful for one of the following reasons: • The initial character is a sharp sign, but the string is not in the proper format. • The initial character is not a sharp sign and the color does not exist in the database of the server.

Error Codes

BadColor

BadImplementation

Implementation Specifics

This Xlib subroutine is part of AIXwindows Run Time Environment in AIXwindows Environment/6000.

Related Information

The **XAllocColor** subroutine, **XAllocNamedColor** subroutine, **XStoreColor** subroutine.

The **LookupColor** protocol request.

XParseGeometry Subroutine

Purpose

Parses standard window geometry.

Libraries

Enhanced X-Windows Library (**libX11.a**)

FORTRAN 77 Library (**libXfx.a**)

C Syntax

int XParseGeometry(*ParseString*, *XReturn*, *YReturn*, *WidthReturn*,*HeightReturn*)
**char ****ParseString*;
**int ****XReturn*, ******YReturn*;
**int ****WidthReturn*, ******HeightReturn*;

FORTRAN Syntax

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*)

Description

The **XParseGeometry** subroutine parses standard window geometry. It uses a standard string to indicate window size and placement. Strings to be parsed are in the following format:

```
=||<width>x<height>||{+-}<xoffset>{+-}<yoffset>|
```

Items enclosed in <> are integers; items enclosed in || are optional; and, items enclosed in {} indicate *choose one of*. Brackets should not be displayed in the actual string.

The items in this form map into the parameters of the **XParseGeometry** subroutine.

The **XParseGeometry** subroutine returns a bit mask indicating values (width, height, x offset, and y offset) found in the string. It also indicates if x and y are negative. By convention, the value of –0 is not equal to the value of +0, so that the window can be positioned relative to the right edge or to the bottom edge.

For each value found, the corresponding parameter is updated. For each value not found, the parameter is left unchanged.

Each value is set when it is defined or when one of the signs is set. The bits are represented by the **XValue**, **YValue**, **WidthValue**, **HeightValue**, **XNegative**, or **YNegative** values in the **<X11/Xutil.h>** data file.

If the subroutine returns to the value of **XValue** or **YValue**, the window can be placed at the requested position. The window is not automatically placed; the user must place the window at the requested position.

Parameters

HeightReturn	Returns the height determined.
ParseString	Specifies the string to be parsed.
WidthReturn	Returns the width determined.
XReturn	Returns the x coordinate offset.
YReturn	Returns the y coordinate offset.

Return Value

Bit mask specifying the fields set.

Error Code

BadImplementation

Implementation Specifics

This Xlib subroutine is part of AIXwindows Run Time Environment in AIXwindows Environment/6000.

XPeekEvent Subroutine

Purpose

Peeks at the event queue.

Libraries

Enhanced X-Windows Library (**libX11.a**)

FORTRAN 77 Library (**libXfx.a**)

C Syntax

XPeekEvent(*DisplayPtr*, *EventReturn*)
Display **DisplayPtr*
XEvent **EventReturn*;

FORTRAN Syntax

external fxpeekevent
integer*4 *DisplayPtr*
integer*4 *EventReturn*
call fxpeekevent(*DisplayPtr*, *EventReturn*)

Description

The **XPeekEvent** subroutine returns the first event from the event queue without removing it from the queue. If the queue is empty, this subroutine flushes the output buffer, then blocks until an event is received. The **XPeekEvent** subroutine copies the event into the client-supplied **XEvent** data structure without removing the event from the event queue.

Parameters

DisplayPtr	Specifies the connection to the X Server.
EventReturn	Returns a copy of the matched event's associated structure.

Error Code

BadImplementation

Implementation Specifics

This Xlib subroutine is part of AIXwindows Run Time Environment in AIXwindows Environment/6000.

XPeekIfEvent Subroutine

Purpose

Checks the event queue for a specified matching event without removing it from the queue.

Libraries

Enhanced X-Windows Library (**libX11.a**)

FORTRAN 77 Library (**libXfx.a**)

C Syntax

XPeekIfEvent(*DisplayPtr*, *EventReturn*, *Predicate*, *Argument*)
Display **DisplayPtr*;
XEvent **EventReturn*;
Bool (**Predicate*)();
char **Argument*;

FORTRAN Syntax

external fxpeekifevent
integer*4 *DisplayPtr*
integer*4 *EventReturn*
integer*4 *Predicate*
character*256 *Argument*
call fxpeekifevent(*DisplayPtr*, *EventReturn*, *Predicate*, *Argument*)

Description

The **XPeekIfEvent** subroutine requires a predicate procedure to check the event queue for a matching event. It returns only when the specified predicate procedure returns **True** for an event.

The following predicate procedure is used:

Bool predicate(*DisplayPtr*, *Event*, *Argument*)
Display **DisplayPtr*;
XEvent **Event*;
char **Argument;*

The predicate procedure is called once for each event in the queue until it finds a match. After finding a match, it returns to the value of **True**. If it does not find a match, it returns to the value of **False**.

After the predicate procedure finds a match, the **XPeekIfEvent** subroutine copies the matched event into the client-supplied **XEvent** data structure without removing the event from the queue. The **XPeekIfEvent** subroutine flushes the output buffer if it blocks waiting for additional events.

Parameters

Argument	Specifies the user-supplied value for the *Argument* parameter of the predicate procedure.
DisplayPtr	Specifies the connection to the X Server.
EventReturn	Returns a copy of the associated structure of the matched event.
Predicate	Specifies the predicate procedure to be called.

Error Code

BadImplementation

Implementation Specifics

This Xlib subroutine is part of AIXwindows Run Time Environment in AIXwindows Environment/6000.

XPending Subroutine

Purpose

Gets the number of events that are pending in the event queue.

Libraries

Enhanced X-Windows Library (**libX11.a**)

FORTRAN 77 Library (**libXfx.a**)

C Syntax

int XPending(*DisplayPtr***)**
**Display ****DisplayPtr*;

FORTRAN Syntax

integer*4 fxpending
external fxpending
integer*4 *DisplayPtr*
integer*4 *NumberEvents*
NumberEvents = **fxpending(***DisplayPtr***)**

Description

The **XPending** subroutine returns the number of events received from the X Server but not yet removed from the event queue. Use the **XNextEvent** subroutine or the **XWindowEvent** subroutine to remove events from the queue.

Using the **XPending** subroutine is equivalent to using the **XEventsQueued** subroutine with a *Mode* parameter of **QueuedAfterFlush**.

Parameter

DisplayPtr Specifies the connection to the X Server.

Return Value

Number of events received from the server but not yet removed from the event queue.

Error Code

BadImplementation

Implementation Specifics

This Xlib subroutine is part of AIXwindows Run Time Environment in AIXwindows Environment/6000.

Related Information

The **XNextEvent** subroutine.

XWindowEvent subroutine, **XEventsQueued** subroutine.

Xpermalloc Subroutine

Purpose

Provides for a permanent allocation of memory.

Libraries

Enhanced X-Windows Library (**libX11.a**)

FORTRAN 77 Library (**libXfx.a**)

C Syntax

char *Xpermalloc(*Size***)**
unsigned int *Size*;

FORTRAN Syntax

integer*4 fxpermalloc
external fxpermalloc
integer*4 *Size*
integer*4 *ReturnCode*
ReturnCode = **fxpermalloc(***Size***)**

Description

The **Xpermalloc** subroutine creates a permanent allocation of memory. This subroutine is used by some toolkits to improve performance and storage use by comparison with using the completely general memory allocator.

Parameter

Size	Amount of memory, in bytes, to allocate.

Error Code

BadImplementation

Implementation Specifics

This Xlib subroutine is part of AIXwindows Run Time Environment in AIXwindows Environment/6000.

XPointInRegion Subroutine

Purpose

Determines if a point lies in a specified region.

Libraries

Enhanced X-Windows Library (**libX11.a**)

FORTRAN 77 Library (**libXfx.a**)

C Syntax

int XPointInRegion(*RegionPtr*, *X*, *Y*)
Region *RegionPtr*;
int *X*, *Y*;

FORTRAN Syntax

integer*4 fxpointinregion
external fxpointinregion
integer*4 *RegionPtr*, *X*, *Y*
integer*4 *ReturnCode*
ReturnCode = **fxpointinregion(***RegionPtr*, *X*, *Y*)

Description

The **XPointInRegion** subroutine determines if a specified point lies in a specified region. The values of the *X* and *Y* parameters define the coordinates of the point.

Parameters

RegionPtr	Specifies the region.
X	Specifies the x coordinate of the point.
Y	Specifies the y coordinate of the point.

Implementation Specifics

This Xlib subroutine is part of AIXwindows Run Time Environment in AIXwindows Environment/6000.

Return Values

False	The point defined by the x and y coordinates does not lie in the specified region.
True	The point defined by the x and y coordinates lies in the specified region.

XPolygonRegion Subroutine

Purpose

Generates a region from a polygon.

Libraries

Enhanced X-Windows Library (**libX11.a**)

FORTRAN 77 Library (**libXfx.a**)

C Syntax

Region XPolygonRegion(*Points*, *Number*, *FillRule***)**
XPoint *Points*[];
int *Number*;
int *FillRule*;

FORTRAN Syntax

integer*4 fxpolygonregion
external fxpolygonregion
integer*4 *Number*
integer*4 *Points*, *FillRule*
integer*4 *Region*
Region = **fxpolygonregion(***Points*, *Number*, *FillRule***)**

Description

The **XPolygonRegion** subroutine returns a region for a polygon defined by an array of points.

The *FillRule* parameter can be set to either the value of **EvenOddRule** or **WindingRule**.

Parameters

FillRule	Specifies the fill rule to be set for the specified graphics context.
Number	Specifies the number of points in the polygon.
Points	Specifies an array of points.

Error Code

BadImplementation

Implementation Specifics

This Xlib subroutine is part of AIXwindows Run Time Environment in AIXwindows Environment/6000.

Related Information

The **XCreateGC** subroutine.

Using Enhanced X-Windows to Draw Points, Lines, Rectangles, and Arcs

XPutBackEvent Subroutine

Purpose

Pushes an event back into the event queue.

Libraries

Enhanced X-Windows Library (**libX11.a**)

FORTRAN 77 Library (**libXfx.a**)

C Syntax

XPutBackEvent(*DisplayPtr*, *Event***)**
Display ***DisplayPtr*;**
XEvent ***Event*;**

FORTRAN Syntax

external fxputbackevent
integer*4 *DisplayPtr*
integer*4 *Event*
call fxputbackevent(*DisplayPtr*, *Event***)**

Description

The **XPutBackEvent** subroutine pushes an event back to the top of the event queue of the current display. When using this subroutine an event can be read, then dealt with later. There is no limit to the number of times in succession the **XPutBackEvent** subroutine can be called.

Parameters

DisplayPtr	Specifies the connection to the X Server.
Event	Specifies a pointer to an event.

Error Code

BadImplementation

Implementation Specifics

This Xlib subroutine is part of AIXwindows Run Time Environment in AIXwindows Environment/6000.

XPutImage Subroutine

Purpose

Combines an image in memory with a rectangle of a drawable on the display screen.

Libraries

Enhanced X-Windows Library (**libX11.a**)

FORTRAN 77 Library (**libXfx.a**)

C Syntax

XPutImage(*DisplayPtr*, *DrawableID*, *GraphicsContext*, *Image*, *SourceX SourceY*, *DestinationX*, *DestinationY*, *Width*, *Height*)
Display **DisplayPtr*;
Drawable *DrawableID*;
GC *GraphicsContext*;
XImage **Image*;
int *SourceX*, *SourceY*;
int *DestinationX*, *DestinationY*;
unsigned int *Width*, *Height*.

FORTRAN Syntax

external fxputimage
integer*4 *DisplayPtr*
integer*4 *DrawableID*, *GraphicsContext*, *Image*
integer*4 *SourceX*, *SourceY*, *DestinationX*, *DestinationY*
integer*4 *Width*, *Height*
call fxputimage(*DisplayPtr*, *DrawableID*, *GraphicContext*, *Image*, *SourceX*, *SourceY*, *DestinationX*, *DestinationY*, *Width*, *Height*)

Description

The **XPutImage** subroutine combines an image in memory with a rectangle of a specified drawable.

If the **XYBitmap** format is used, the depth of the image must be a value of 1. The foreground pixel in the graphics context defines the source for the one bits in the image, and the background pixel defines the source for the 0–bits.

If the **XYPixmap** and **ZPixmap** values are used, the depth of the image must match the depth of the drawable. The section of the image defined by the *SourceX*, *SourceY*, *Width*, and *Height* parameters is drawn on the specified part of the drawable.

The **XPutImage** subroutine uses the *function*, *plane_mask*, *subwindow_mode*, *clip_x_origin*, *clip_y_origin*, and *clip_mask* graphics context fields. It also uses the *foreground* and *background* graphics context mode-dependent fields.

Parameters

DestinationX	Specifies the x coordinate of the subimage, relative to the origin of the drawable.
DestinationY	Specifies the y coordinate of the subimage, relative to the origin of the drawable.

DisplayPtr	Specifies the connection to the X Server.
DrawableID	Specifies the resource ID.
GraphicContext	Specifies the graphics context.
Height	Specifies the height of the subimage.
Image	Specifies the image to be combined with the rectangle.
SourceX	Specifies the offset in the x coordinate from the left edge of the image defined by the **XImage** data structure.
SourceY	Specifies the offset in the y coordinate from the top edge of the image defined by the **XImage** data structure.
Width	Specifies the width of the subimage.

Error Codes

BadDrawable

BadGC

BadImplementation

BadMatch

BadValue

Implementation Specifics

This Xlib subroutine is part of AIXwindows Run Time Environment in AIXwindows Environment/6000.

Related Information

The **PutImage** protocol request.

XPutPixel Subroutine

Purpose

Sets a pixel value in an image.

Libraries

Enhanced X-Windows Library (**libX11.a**)

FORTRAN 77 Library (**libXfx.a**)

C Syntax

int XPutPixel(*XImagePtr*, *X*, *Y*, *Pixel***)**
XImage **XImagePtr*;
int *X*;
int *Y*;
unsigned long *Pixel*;

FORTRAN Syntax

integer*4 fxputpixel
external fxputpixel
integer*4 *XImagePtr*, *X*, *Y*, *Pixel*
integer*4 *Status*
Status = **fxputpixel(***XImagePtr*, *X*, *Y*, *Pixel***)**

Description

The **XPutPixel** subroutine sets a pixel value in an image, overwriting the pixel value in the specified image with a new value.

The input pixel value must be in normalized format. The least-significant byte of the long flag defined in the **<X11/Xutil.h>** data file is the least-significant byte of the pixel.

The image must contain the x and y coordinates.

Parameters

Pixel	Specifies the new pixel value.
X	Specifies the x coordinate relative to the origin of the image.
XImagePtr	Specifies a pointer to the image.
Y	Specifies the y coordinate relative to the origin of the image.

Error Code

BadImplementation

Implementation Specifics

This Xlib subroutine is part of AIXwindows Run Time Environment in AIXwindows Environment/6000.

XQueryBestCursor Subroutine

Purpose

Gets the best size for the cursor.

Libraries

Enhanced X-Windows Library (**libX11.a**)

FORTRAN 77 Library (**libXfx.a**)

C Syntax

Status XQueryBestCursor(*DisplayPtr*, *DrawableID*, *Width*, *Height*,*WidthReturn*, *HeightReturn*)
Display **DisplayPtr*;
Drawable *DrawableID*;
unsigned int *Width*, *Height*;
unsigned int **WidthReturn*, **HeightReturn*;

FORTRAN Syntax

integer*4 fxquerybestcursor
external fxquerybestcursor
integer*4 *DisplayPtr*
integer*4 *DrawableID*, *Width*, *Height*
integer*4 *WidthReturn*, *HeightReturn*
integer*4 *Status*
Status = **fxquerybestcursor**(*DisplayPtr*, *DrawableID*, *Width*, *Height*, *WidthReturn*, *HeightReturn*)

Description

The **XQueryBestCursor** subroutine provides a way to find out what size cursors are actually possible on the display. It returns the largest size that can be displayed. Applications should be prepared to use smaller cursors on displays that cannot support large ones.

Parameters

DisplayPtr	Specifies the connection to the X Server.
DrawableID	Specifies any drawable.
Height	Specifies the height of the cursor.
HeightReturn	Returns the height dimension closest to the specified height.
Width	Specifies the width of the cursor.
WidthReturn	Returns the width dimension closest to the specified width.

Return Values

False	The **XQueryBestCursor** subroutine is not successful.
True	The **XQueryBestCursor** subroutine is successful.

Error Codes

BadDrawable

BadImplementation

Implementation Specifics

This Xlib subroutine is part of AIXwindows Run Time Environment in AIXwindows Environment/6000.

Related Information

The **QueryBestSize** protocol request.

XQueryBestSize Subroutine

Purpose

Gets the best size for the tile, stipple, or cursor.

Libraries

Enhanced X-Windows Library (**libX11.a**)

FORTRAN 77 Library (**libXfx.a**)

C Syntax

Status XQueryBestSize(*DisplayPtr*, *Class*, *WhichScreen*, *Width*,, *Height*,*WidthReturn* , *HeightReturn*)
Display **DisplayPtr*;
int *Class*;
Drawable *WhichScreen*;
unsigned int *Width*, *Height*;
unsigned int **WidthReturn*, **HeightReturn*;

FORTRAN Syntax

integer*4 fxquerybestsize
external fxquerybestsize
integer*4 *DisplayPtr*
integer*4 *Class*
integer*4 *WhichScreen*
integer*4 *Width*, *Height*
integer*4 *WidthReturn*, *HeightReturn*
integer*4 *Status*
Status = **fxquerybestsize**(*DisplayPtr*, *Class*, *WhichScreen*, *Width*, *Height*, *WidthReturn*, *HeightReturn*)

Description

The **XQueryBestSize** subroutine returns the best size of a tile, stipple, or cursor.

If the *Class* parameter is specified as the value of **CursorShape**, the **XQueryBestSize** subroutine returns the largest size that can be fully displayed on the display screen specified in the *WhichScreen* parameter.

If the *Class* parameter is specified as the value of **TileShape**, the **XQueryBestSize** subroutine returns the size that can be tiled fastest. The drawable specified in the *WhichScreen* parameter indicates the display screen and, optionally, the window class and depth.

If the *Class* parameter is specified as the value of **StippleShape**, the **XQueryBestSize** subroutine returns the size that can be stippled fastest.

An **InputOnly** window cannot be used as the drawable if the *Class* parameter is specified as the value of **TileShape** or **StippleShape**.

The drawable specified in the *WhichScreen* parameter indicates the display screen and, optionally, the window class and depth.

Parameters

Class	Specifies the class as the value of **TileShape**, **CursorShape**, or **StippleShape**.
DisplayPtr	Specifies the connection to the X Server.
Height	Specifies the height of the drawable.
HeightReturn	Returns the height of the drawable best supported by the display system.
WhichScreen	Specifies any drawable on the screen.
Width	Specifies the width of the drawable.
WidthReturn	Returns the width of the drawable best supported by the display system.

Return Values

False	The **XQueryBestSize** subroutine is not successful.
True	The **XQueryBestSize** subroutine is successful.

Error Codes

BadDrawable

BadImplementation

BadMatch

BadValue

Implementation Specifics

This Xlib subroutine is part of AIXwindows Run Time Environment in AIXwindows Environment/6000.

XQueryBestStipple Subroutine

Purpose

Gets best stipple shape.

Libraries

Enhanced X-Windows Library (**libX11.a**)

FORTRAN 77 Library (**libXfx.a**)

C Syntax

Status XQueryBestStipple(*DisplayPtr*, *WhichScreen*, *Width*, *Height*, *WidthReturn*
,*HeightReturn*)
Display **DisplayPtr*;
Drawable *WhichScreen*;
unsigned int *Width*, *Height*;
unsigned int **WidthReturn*, **HeightReturn*;

FORTRAN Syntax

integer*4 fxquerybeststipple
external fxquerybeststipple
integer*4 *DisplayPtr*
integer*4 *WhichScreen*
integer*4 *Width*, *Height*
integer*4 *WidthReturn*, *HeightReturn*
integer*4 *Status*
Status = **fxquerybeststipple(***DisplayPtr*, *WhichScreen*, *Width*, *Height*, *WidthReturn*,
HeightReturn)

Description

The **XQueryBestStipple** subroutine obtains the size that can be stippled fastest on the screen, closest to the size specified.

The drawable specified by the *WhichScreen* parameter indicates the display screen and, optionally, the window class and depth.

An **InputOnly** window cannot be used as the drawable for this subroutine.

Parameters

DisplayPtr	Specifies the connection to the X Server.
Height	Specifies the height of the drawable.
HeightReturn	Returns the height best supported by the display system.
WhichScreen	Specifies any drawable.
Width	Specifies the width of the drawable.
WidthReturn	Returns the width best supported by the display system.

Return Values

False The **XQueryBestStipple** subroutine is not successful.

True The **XQueryBestStipple** subroutine is successful.

Error Codes

BadDrawable

BadImplementation

BadMatch

Implementation Specifics

This Xlib subroutine is part of AIXwindows Run Time Environment in AIXwindows Environment/6000.

Related Information

The **QueryBestSize** protocol request.

XQueryBestTile Subroutine

Purpose

Gets the best fill tile shape.

Libraries

Enhanced X-Windows Library (**libX11.a**)

FORTRAN 77 Library (**libXfx.a**)

C Syntax

Status XQueryBestTile(*DisplayPtr*, *WhichScreen*, *Width*, *Height*, *WidthReturn* , *HeightReturn*)
Display **DisplayPtr*;
Drawable *WhichScreen*;
unsigned int *Width*, *Height*;
unsigned int **WidthReturn*, **HeightReturn*;

FORTRAN Syntax

integer*4 fxquerybesttile
external fxquerybesttile
integer*4 *DisplayPtr*
integer*4 *WhichScreen*
integer*4 *Width*, *Height*
integer*4 *WidthReturn*, *HeightReturn*
integer*4 *Status*
Status = **fxquerybesttile(***DisplayPtr*, *WhichScreen*, *Width*, *Height*, *WidthReturn*, *HeightReturn*)

Description

The **XQueryBestTile** subroutine obtains the size that can be tiled fastest on the screen, closest to the size specified.

The drawable specified in the *WhichScreen* parameter indicates the display screen and, optionally, the window class and depth.

An **InputOnly** window cannot be used as the drawable for this subroutine.

Parameters

DisplayPtr	Specifies the connection to the X Server.
Height	Specifies the height of the drawable.
HeightReturn	Returns the height best supported by the display system.
WhichScreen	Specifies any drawable.
Width	Specifies the width of the drawable.
WidthReturn	Returns the width best supported by the display system.

Return Values

False	The **XQueryBestTile** subroutine is not successful.
True	The **XQueryBestTile** subroutine is successful.

Error Codes

BadDrawable

BadImplementation

BadMatch

Implementation Specifics

This Xlib subroutine is part of AIXwindows Run Time Environment in AIXwindows Environment/6000.

Related Information

The **QueryBestSize** protocol request.

XQueryColor Subroutine

Purpose

Obtains the RGB (red, green, and blue) value for a specified pixel.

Libraries

Enhanced X-Windows Library (**libX11.a**)

FORTRAN 77 Library (**libXfx.a**)

C Syntax

XQueryColor(*DisplayPtr*, *ColormapID*, *DefinitionInOut***)**
Display **DisplayPtr*;
Colormap *ColormapID*;
XColor **DefinitionInOut*;

FORTRAN Syntax

external fxquerycolor
integer*4 *DisplayPtr*
integer*4 *ColormapID*
integer*4 *DefinitionInOut*
call fxquerycolor(*DisplayPtr*, *ColormapID*, *DefinitionInOut***)**

Description

The **XQueryColor** subroutine obtains the color values for a single specified pixel value. It returns the RGB values stored in the *Colormap* parameter for the pixel value passed as the **pixel** field in the **XColor** data structure. This subroutine sets the **flags** field of the **XColor** data structure to the appropriate flag.

Parameters

ColormapID	Specifies the colormap ID.
DefinitionInOut	Returns the RGB value. Specifies the pixel value. This is both the Input and the Output parameter.
DisplayPtr	Specifies the connection to the X Server.

Error Codes

BadColor

BadImplementation

BadValue

Implementation Specifics

This Xlib subroutine is part of AIXwindows Run Time Environment in AIXwindows Environment/6000.

Related Information

The **QueryColors** protocol request.

XQueryColors Subroutine

Purpose

Queries the RGB (red, green, and blue) values for an array of pixels.

Libraries

Enhanced X-Windows Library (**libX11.a**)

FORTRAN 77 Library (**libXfx.a**)

C Syntax

XQueryColors(*DisplayPtr*, *ColormapID*, *DefinitionsInOut*, *NumberColors***)**
Display **DisplayPtr*;
Colormap *ColormapID*;
XColor *DefinitionsInOut[]*;
int *NumberColors*;

FORTRAN Syntax

external fxquerycolor
integer*4 *DisplayPtr*
integer*4 *ColormapID*
integer*4 *DefinitionsInOut*
integer*4 *NumberColors*
call fxquerycolor(*DisplayPtr*, *ColormapID*, *DefinitionsInOut***)**

Description

The **XQueryColors** subroutine obtains color values for a list of pixels stored in the list of **XColor** data structures. It returns the RGB values stored in the *Colormap* parameter for the pixel value passed in the **pixel** fields of **XColor** data structures. The **XQueryColors** sets the **flags** field of the **XColor** data structure to the appropriate flags.

Parameters

ColormapID	Specifies the colormap ID.
DefintionsInOut	Specifies pixels and returns a list of RGB color definition data structures for the pixels specified in the data structure.
DisplayPtr	Specifies the connection to the X Server.
NumberColors	Specifies the number of **XColor** data structures in the color definition list.

Error Codes

BadColor

BadImplementation

BadValue

Implementation Specifics

This Xlib subroutine is part of AIXwindows Run Time Environment in AIXwindows Environment/6000.

Related Information

The **QueryColors** protocol request.

XQueryFont Subroutine

Purpose

Returns information about a loaded font.

Libraries

Enhanced X-Windows Library (**libX11.a**)

FORTRAN 77 Library (**libXfx.a**)

C Syntax

XFontStruct *XQueryFont(*DisplayPtr*, *FontID***)**
**Display ***DisplayPtr*;
XID *FontID*;

FORTRAN Syntax

integer*4 fxqueryfont
external fxqueryfont
integer*4 *DisplayPtr*
integer*4 *FontID*
integer*4 *FontID*
FontID = **fxqueryfont(***DisplayPtr*, *FontID***)**

Description

The **XQueryFont** subroutine returns information about a loaded font and returns a pointer to the **XFontStruct** data structure, which provides information about the font.

The **XQueryFont** subroutine can query a font or the fonts stored in the graphics context. The font ID stored in the **XFontStruct** data structure will be the **GContext** ID. (The **GContext** ID, however, is not valid as a font ID in all other subroutines.)

The **XFreeFontInfo** subroutine frees the data obtained by using the **XQueryFont** subroutine.

Parameters

DisplayPtr Specifies the connection to the X Server.

FontID Specifies the font ID or the **GContext** ID.

Return Values

NULL The query about a font is not successful.

Pointer to the **XFontStruct** data structure containing the information.

Error Code

BadImplementation

Implementation Specifics

This Xlib subroutine is part of AIXwindows Run Time Environment in AIXwindows Environment/6000.

Related Information

The **XFreeFont** subroutine, **XFreeFontInfo** subroutine, **XGContextFromGC** subroutine, **XLoadQueryFont** subroutine, **XListFonts** subroutine, **XListFontsWithInfo** subroutine.

The **QueryFont** protocol request.

XQueryKeymap Subroutine

Purpose

Gets a bit vector that describes the state of the keyboard.

Libraries

Enhanced X-Windows Library (**libX11.a**)

FORTRAN 77 Library (**libXfx.a**)

C Syntax

XQueryKeymap(*DisplayPtr*, *KeysReturn*)
Display **DisplayPtr*;
char *KeysReturn*[**32**];

FORTRAN Syntax

external fxquerykeymap
integer*4 *DisplayPtr*
integer*4 *KeysReturn*
call fxquerykeymap(*DisplayPtr*, *KeysReturn*)

Description

The **XQueryKeymap** subroutine returns a bit vector representing the logical state of the keyboard, where each 1-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`.

Parameters

DisplayPtr	Specifies the connection to the X Server.
KeysReturn	Returns an array of bytes, where each bit represents one key of the keyboard, to identify which keys are pressed down.

Error Code

BadImplementation

Implementation Specifics

This Xlib subroutine is part of AIXwindows Run Time Environment in AIXwindows Environment/6000.

Related Information

The **QueryKeymap** protocol request.

XQueryPointer Subroutine

Purpose

Obtains the root window and the pointer coordinates relative to the origin of the root for the current pointer position.

Libraries

Enhanced X-Windows Library (**libX11.a**)

FORTRAN 77 Library (**libXfx.a**)

C Syntax

Bool XQueryPointer(*DisplayPtr*, *WindowID*, *RootReturn*, *ChildReturn*, *RootXReturn*, *RootYReturn*, *WindowXReturn*, *WindowYReturn*, *MaskReturn*)
Display **DisplayPtr*;
Window *WindowID*;
Window **RootReturn*, **ChildReturn*;
int **RootXReturn*, **RootYReturn*;
int **WindowXReturn*, **WindowYReturn*;
unsigned int **MaskReturn*;

FORTRAN Syntax

integer*4 fxquerypointer
external fxquerypointer
integer*4 *DisplayPtr*
integer*4 *WindowID*
integer*4 *RootReturn*, *ChildReturn*
integer*4 *RootXReturn*, *RootYReturn*
integer*4 *WindowXReturn*, *WindowYReturn*
integer*4 *MaskReturn*
integer*4 *ReturnCode*
ReturnCode = **fxquerypointer(***DisplayPtr*, *WindowID*, *RootReturn*, *ChildReturn*, *RootXReturn*, *RootYReturn*, *WindowXReturn*, *WindowYReturn*, *MaskReturn*)

Description

The **XQueryPointer** subroutine returns the root window and the pointer coordinates relative to the origin of the root for the current pointer position.

If the pointer coordinates returned to the *WindowXReturn* and *WindowYReturn* parameters are relative to the origin of the specified window, the **XQueryPointer** subroutine returns **True**. If a child window contains the pointer, its window ID is returned in the *ChildReturn* parameter. Otherwise, the *ChildReturn* parameter is **None**.

If the pointer is not on the same screen as the specified window, the **XQueryPointer** subroutine returns **False**. In addition, **None** is returned in the *ChildReturn* parameter, and value of 0 is returned in the *WindowXReturn* and *WindowYReturn* parameters.

The **XQueryPointer** subroutine returns the current logical state of the pointer buttons and modifier keys to the *MaskReturn* parameter. It sets the *MaskReturn* parameter to the bitwise-inclusive OR of the current state of one or more of the pointer button or modifier key bit masks.

Parameters

ChildReturn	Returns the child window ID, if any, for the current pointer position.
DisplayPtr	Specifies the connection to the X Server.
MaskReturn	Returns the current state of the modifier keys and pointer buttons.
RootReturn	Returns the root window ID for the current pointer position.
RootXReturn	Returns the x coordinate for the current pointer position, relative to the origin of the root window.
RootYReturn	Returns the y coordinate for the current pointer position, relative to the origin of the root window.
WindowID	Specifies the window ID.
WindowXReturn	Returns the x coordinate for the current pointer position, relative to the origin of a specified window.
WindowYReturn	Returns the y coordinate for the current pointer position, relative to the origin of a specified window.

Return Values

True	The pointer coordinates returned to the *WindowXReturn* and *WindowYReturn* parameters are relative to the origin of the specified window.
	Returns the Child ID of the window containing the pointer, if any.
False	The pointer is not on the same screen as the specified window.
	Returns the value of **None** to *ChildReturn*.
	Returns **0** to the *WindowXReturn* and *WindowYReturn* parameters.

Error Codes

BadImplementation

BadWindow

Implementation Specifics

This Xlib subroutine is part of AIXwindows Run Time Environment in AIXwindows Environment/6000.

Related Information

The **QueryPointer** protocol request.

XQueryTextExtents Subroutine

Purpose

Queries the server for the bounding box of an 8-bit character string in a specified font.

Libraries

Enhanced X-Windows Library (**libX11.a**)

FORTRAN 77 Library (**libXfx.a**)

C Syntax

XQueryTextExtents(*DisplayPtr*, *FontID*, *String*, *NumberCharacters*, *DirectionReturn*, *FontAscentReturn*, *FontDescentReturn*, *OverallReturn*)
Display **DisplayPtr*;
XID *FontID*;
XChar2b **String*;
int *NumberCharacters*;
int **DirectionReturn*;
int **FontAscentReturn*, **FontDescentReturn*;
XCharStruct **OverallReturn*;

FORTRAN Syntax

external fxquerytextextents
integer*4 *DisplayPtr*
integer*4 *FontID*
character*256 *String*
integer*4 *NumberCharacters*, *DirectionReturn*, *FontAscentReturn*
integer*4 *FontDescentReturn*, *OverallReturn*
call fxquerytextextents(*DisplayPtr*, *FontID*, *String*, *NumberCharacters*, *DirectionReturn*, *FontAscentReturn*, *FontDescentReturn*, *OverallReturn*)

Description

The **XQueryTextExtents** subroutine returns either the bounding box of a specified 8-bit character string in a font or the font contained in a specified graphics context. It returns an **XCharStruct** data structure with the following values:

- The ascent member is set to the maximum of the ascent metrics of all characters in the string.
- The descent member is set to the maximum of the descent metrics.
- The width member is set to the sum of the character-width metrics of all characters in the string.
- Let `W` be the sum of the character-width metrics of all characters preceding it for each character in the string.
- Let `L` be the left-side-bearing metric of the character plus `W`.
- Let `R` be the right-side-bearing metric of the character plus `W`.
- The *lbearing* field is set to the minimum `L` of all characters in the string.
- The *rbearing* field is set to the maximum `R` of all characters in the string.

If the font has no defined default character, undefined characters in the string are taken to have zero metrics.

Note: Since the **XQueryTextExtents** subroutine queries the X Server, there is more round-trip overhead involved than when using the **XTextExtents** subroutine.

Parameters

DirectionReturn	Returns the value of the direction hint field, which can be either the value of **FontLeftToRight** or **FontRightToLeft**.
DisplayPtr	Specifies the connection to the X Server.
FontAscentReturn	Returns the font ascent member.
FontDescentReturn	Returns the font descent member.
FontID	Specifies either the font ID or the **GContext** value ID that contains the font.
NumberCharacters	Specifies the number of characters in the string.
OverallReturn	Returns the overall size in the specified **XCharStruct** data structure.
String	Specifies a character string.

Error Codes

BadFont

BadGC

BadImplementation

Implementation Specifics

This Xlib subroutine is part of AIXwindows Run Time Environment in AIXwindows Environment/6000.

Related Information

The **XCharStruct** data structure.

The **QueryTextExtents** protocol request.

XQueryTextExtents16 Subroutine

Purpose

Queries the server for the bounding box of a 2-byte, 16-bit, character string in a specified font.

Libraries

Enhanced X-Windows Library (**libX11.a**)

FORTRAN 77 Library (**libXfx.a**)

C Syntax

XQueryTextExtents16(*DisplayPtr*, *FontID*, *String*, *NumberCharacters*, *DirectionReturn*, *FontAscentReturn*, *FontDescentReturn*, *OverallReturn*)
Display **DisplayPtr*;
XID *FontID*;
XChar2b **String*;
int *NumberCharacters*;
int **DirectionReturn*;
int **FontAscentReturn*, **FontDescentReturn*;
XCharStruct **OverallReturn*;

FORTRAN Syntax

external fxquerytextextents16
integer*4 *DisplayPtr*
integer*4 *FontID*
integer*4 *String*
integer*4 *NumberCharacters*, *DirectionReturn*, *FontAscentReturn*
integer*4 *FontDescentReturn*, *OverallReturn*
call fxquerytextextents16(*DisplayPtr*, *FontID*, *String*, *NumberCharacters*, *DirectionReturn*, *FontAscentReturn*, *FontDescentReturn*, *OverallReturn*)

Description

The **XQueryTextExtents16** subroutine returns either the bounding box of a specified 16-bit character string in a font or the font contained in a specified graphics context. It returns an **XCharStruct** data structure with the following values:

- The ascent member is set to the maximum of the ascent metrics of all characters in the string.
- The descent member is set to the maximum of the descent metrics.
- The width member is set to the sum of the character-width metrics of all characters in the string.
- Let `W` be the sum of the character-width metrics of all characters preceding it for each character in the string.
- Let `L` be the left-side-bearing metric of the character plus `W`.
- Let `R` be the right-side-bearing metric of the character plus `W`.
- The *lbearing* field is set to the minimum `L` of all characters in the string.
- The *rbearing* field is set to the maximum `R` of all characters in the string.

For fonts defined with linear indexing rather than 2-byte matrix indexing, each **XChar2b** data structure is interpreted as a 16-bit number with *Byte1* as the most-significant byte. If the font has no defined default character, undefined characters in the string are taken to have zero metrics.

Note: Since the **XQueryTextExtents16** subroutine queries the X Server, there is more round-trip overhead involved than when using the **XTextExtents16**.

Parameters

FontAscentReturn	Returns the font ascent member.
FontDescentReturn	Returns the font descent member.
DirectionReturn	Returns the value of the direction hint field which can be either the value of **FontLeftToRight** or **FontRightToLeft**.
DisplayPtr	Specifies the connection to the X Server.
FontID	Specifies either the font ID or the **GContext** value ID that contains the font.
NumberCharacters	Specifies the number of characters in the string.
OverallReturn	Returns the overall size in the specified **XCharStruct** data structure.
String	Specifies a character string.

Error Codes

BadFont

BadGC

BadImplementation

Implementation Specifics

This Xlib subroutine is part of AIXwindows Run Time Environment in AIXwindows Environment/6000.

Related Information

The **XChar2b** data structure.

The **QueryTextExtents** protocol request.

XQueryTree Subroutine

Purpose

Obtains information on the window tree for a specified window.

Libraries

Enhanced X-Windows Library (**libX11.a**)

FORTRAN 77 Library (**libXfx.a**)

C Syntax

Status XQueryTree(*DisplayPtr*, *WindowID*, *RootReturn*, *ParentReturn*, *ChildrenReturn*, *NumberChildrenReturn*)
Display **DisplayPtr*;
Window *WindowID*;
Window **RootReturn*;
Window **ParentReturn*;
Window ***ChildrenReturn*;
unsigned int **NumberChildrenReturn*;

FORTRAN Syntax

integer*4 fxquerytree
external fxquerytree
integer*4 *DisplayPtr*
integer*4 *WindowID*
integer*4 *RootReturn*, *ParentReturn*
integer*4 *ChildrenReturn*
integer*4 *NumberChildrenReturn*
integer*4 *Status*
Status = **fxquerytree**(*DisplayPtr*, *WindowID*, *RootReturn*, *ParentReturn*, *ChildrenReturn*, *NumberChildrenReturn*)

Description

The **XQueryTree** subroutine returns the root ID, the parent window ID, a pointer to the list of child windows, and the number of children for a specified window. The child windows are listed in current stacking order from bottom to top (first to last).

Use the **XFree** subroutine to free this list when it is no longer needed.

Parameters

ChildrenReturn	Returns a pointer to the list of children for a specified window.
DisplayPtr	Specifies the connection to the X Server.
NumberChildrenReturn	Returns the number of children for a specified window.
ParentReturn	Returns the window ID of the parent for a specified window.
RootReturn	Returns the root window ID for a specified window.
WindowID	Specifies the window ID.

Return Values

False The **XQueryTree** subroutine does not succeed.

True The **XQueryTree** subroutine succeeds.

Error Codes

BadImplementation

BadWindow

Implementation Specifics

This Xlib subroutine is part of AIXwindows Run Time Environment in AIXwindows Environment/6000.

Related Information

The **QueryTree** protocol request.

XRaiseWindow Subroutine

Purpose

Raises the specified window.

Library

Enhanced X-Windows Library (**libX11.a**)

FORTRAN 77 Library (**libXfx.a**)

C Syntax

XRaiseWindow(*DisplayPtr*, *WindowID*)
Display **DisplayPtr*;
Window *WindowID*;

FORTRAN Syntax

external fxraisewindow
integer*4 *DisplayPtr*
integer*4 *WindowID*
call fxraisewindow(*DisplayPtr*, *WindowID*)

Description

The **XRaiseWindow** subroutine raises the specified window to the top of the stack so that a sibling window does not obscure it. If the windows are regarded as overlapping sheets of paper stacked on a desk, raising a window is the same as moving the sheet to the top of the stack while leaving its x and y location on the desk constant.

Raising a mapped window may generate the **Expose** events for the window and for any mapped subwindows that were formerly obscured.

If the *override_redirect* field of the window is the value of **False** and some other client has selected the **SubstructureRedirectMask** value on the parent window, the X Server generates a **ConfigureRequest** event, and processing is not performed. Otherwise, the window is raised.

Parameters

DisplayPtr	Specifies the connection to the X Server.
WindowID	Specifies the window ID.

Error Codes

BadWindow

BadImplementation

Implementation Specifics

This Xlib subroutine is part of AIXwindows Run Time Environment in AIXwindows Environment/6000.

Related Information

The **ConfigureWindow** protocol request.

XReadBitmapFile Subroutine

Purpose

Creates a bitmap from a description in a bitmap file.

Libraries

Enhanced X-Windows Library (**libX11.a**)

FORTRAN 77 Library (**libXfx.a**)

C Syntax

int XReadBitmapFile(*DisplayPtr*, *DrawableID*, *Filename*, *WidthReturn*, *HeightReturn*, *BitmapReturn*, *XHotReturn*, *YHotReturn*)
Display **DisplayPtr*;
Drawable *DrawableID*;
char **Filename*;
unsigned int **WidthReturn*, **HeightReturn*;
Pixmap **BitmapReturn*;
int **XHotReturn*, **YHotReturn*;

FORTRAN Syntax

integer*4 fxreadbitmapfile
external fxreadbitmapfile
integer*4 *DisplayPtr*, *DrawableID*
character*256 *Filename*
integer*4 *WidthReturn*, *HeightReturn*, *BitmapReturn*
integer*4 *XHotReturn*, *YHotReturn*
integer*4 *Status*
Status = **fxreadbitmapfile(***DisplayPtr*, *DrawableID*, *Filename*, *WidthReturn*, *HeightReturn*, *BitmapReturn*, *XHotReturn*, *YHotReturn*)

Description

The **XReadBitmapFile** subroutine creates a bitmap from a description in a bitmap file.

The **XReadBitmapFile** subroutine assigns the height and width from the bitmap file that was read to the height and width of the target bitmap file or the file initiating the call. This subroutine then creates a pixmap using the **XCreatePixmap** subroutine, reads the bitmap data from the file into the pixmap, and assigns the pixmap to the bitmap of the target file.

When this subroutine is completed, free the bitmap with the **XFreePixmap** subroutine.

If the x and y hot spots have assigned values, the **XReadBitmapFile** subroutine returns these values. If a hot spot is not defined, the **XReadBitmapFile** subroutine sets the x and y hot spots to the values of –1, –1.

Parameters

DisplayPtr	Specifies the connection to the X Server.
DrawableID	Specifies the drawable.
Filename	Specifies the file name to use. The format of the file name depends on the operating system.

WidthReturn	Returns the width value of the read operation in the bitmap file.
HeightReturn	Returns the height values of the read operation in the bitmap file.
BitmapReturn	Returns the bitmap ID created.
XHotReturn	Returns the x hot spot coordinate.
YHotReturn	Returns the y hot spot coordinates.

Return Values

BitmapOpenFailed	The file cannot be opened.
BitmapFileInvalid	The file can be opened but it contains invalid bitmap data.
BitmapNoMemory	The insufficient working space was allocated.
BitmapSuccess	The file is readable and valid.

Error Codes

BadAlloc

BadImplementation

Implementation Specifics

This Xlib subroutine is part of AIXwindows Run Time Environment in AIXwindows Environment/6000.

Related Information

The **XFreePixmap** subroutine.

XRebindCode Subroutine

Purpose

Changes the keyboard mapping in the key map file.

Libraries

Enhanced X-Windows Library (**liboldX.a**)

FORTRAN 77 Library (**libXfx.a**)

C Syntax

XRebindCode(*DisplayPtr*, *Keycode*, *Shiftbits*, *String*, *NumberBytes*)
Display **DisplayPtr*;
unsigned int *Keycode*;
unsigned int *Shiftbits*;
char **String*;
int *NumberBytes*;

FORTRAN Syntax

external fxrebindcode
integer*4 *DisplayPtr*
integer*4 *Keycode*
integer*4 *ShiftBits*
integer*4 *String*
integer*4 *NumberBytes*
call fxrebindcode(*DisplayPtr*, *Keycode*, *Shiftbits*, *String*, *NumberBytes*)

Description

The **XRebindCode** subroutine changes the binding of the keyboard temporarily. After issuing the **XRebindCode** subroutine, subsequent calls to the **XLookupMapping** subroutine return 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 the *NumberBytes* parameter is a value of 0 and the *String* parameter is not **NULL**, then the *String* parameter points to a 2-byte array that contains the code page and code point of a dead key. If the *String* parameter is the **NULL** value and the *NumberBytes* parameter is not 0, then the *NumberBytes* parameter defines a subroutine ID.

Parameters

DisplayPtr	Specifies the connection to the X Server.
Keycode	Specifies which keycode to change temporarily.
Shiftbits	Specifies shift bits.
String	Returns a pointer to the string.
NumberBytes	Specifies the number of bytes in the string.

Error Code

BadImplementation

Implementation Specifics

This Xlib subroutine is part of AIXwindows Run Time Environment in AIXwindows Environment/6000.

Related Information

The **XLookupMapping** subroutine.

The **keycomp** command.

XRebindKeysym Subroutine

Purpose

Maps character string to the specified key symbol and modifiers.

Libraries

Enhanced X-Windows Library (**libX11.a**)

FORTRAN 77 Library (**libXfx.a**)

C Syntax

XRebindKeysym(*DisplayPtr, KeySym, List, ModifierCount, String, BytesString*)
Display **DisplayPtr*;
KeySym *KeySym*;
KeySym *List[]*;
int *ModifierCount*;
unsigned char **String*;
int *BytesString*;

FORTRAN Syntax

external fxrebindkeysym
integer*4 *DisplayPtr*
integer*4 *KeySym, List, ModifierCount*
character*256 *String*
integer*4 *BytesString*
call fxrebindkeysym(*DisplayPtr, KeySym, List, ModifierCount, String, BytesString*)

Description

The **XRebindKeysym** subroutine changes the bindings of the meaning of a **keysym** for a client. This subroutine does not redefine the key code in the X Server but provides a way to attach long strings to keys. The **XLookupString** subroutine returns this string when the appropriate set of modifier keys is pressed and when the key symbol is used for the translation. You can rebind a key symbol that may not exist.

Parameters

DisplayPtr	Specifies the connection to the AIX Server.
KeySym	Specifies the key symbol to be rebound.
List	Specifies a pointer to an list of key symbols that are being used as modifiers.
ModifierCount	Specifies the number of modifiers in the modifier list.
String	Specifies a pointer to the string to be returned by the **XLookupString** subroutine.
ByteString	Specifies the length of the string.

Error Code

BadImplementation

Implementation Specifics

This Xlib subroutine is part of AIXwindows Run Time Environment in AIXwindows Environment/6000.

Related Information

The **XLookupString** subroutine.

XRecolorCursor Subroutine

Purpose

Changes the color of a cursor.

Libraries

Enhanced X-Windows Library (**libX11.a**)

FORTRAN 77 Library (**libXfx.a**)

C Syntax

XRecolorCursor(*DisplayPtr*, *CursorID*, *ForegroundColor*, *BackgroundColor*)
Display **DisplayPtr*;
Cursor *CursorID*;
XColor **ForegroundColor*, **BackgroundColor*;

FORTRAN Syntax

external fxrecolorcursor
integer*4 *DisplayPtr*
integer*4 *CursorID*
integer*4 *ForegroundColor*, *BackgroundColor*
call fxrecolorcursor(*DisplayPtr*, *CursorID*, *ForegroundColor*, *BackgroundColor*)

Description

The **XRecolorCursor** subroutine changes the color of the specified cursor. If the cursor is being displayed on a screen, this change is visible immediately.

Parameters

DisplayPtr	Specifies the connection to the X Server.
CursorID	Specifies the cursor.
ForegroundColor	Specifies the red, green, and blue (RGB) values for the foreground of the source.
BackgroundColor	Specifies the red, green, and blue (RGB) values for the background of the source.

Error Codes

BadCursor

BadImplementation

Implementation Specifics

This Xlib subroutine is part of AIXwindows Run Time Environment in AIXwindows Environment/6000.

Related Information

The **RecolorCursor** protocol request.

XRectInRegion Subroutine

Purpose

Determines if a rectangle lies in the specified region.

Libraries

Enhanced X-Windows Library (**libX11.a**)

FORTRAN 77 Library (**libXfx.a**)

C Syntax

int XRectInRegion(*RegionPtr*, *X*, *Y*, *Width*, *Height*)
Region *RegionPtr*;
int *X*, *Y*;
unsigned int *Width*, *Height*;

FORTRAN Syntax

integer*4 fxrectinregion
external fxrectinregion
integer*4 *RegionPtr*, *X*, *Y*, *Width*, *Height*
integer*4 *Returncode*
Returncode = **fxrectinregion**(*RegionPtr, X, Y, Width, Height*)

Description

The **XRectInRegion** subroutine determines if a specified rectangle resides in the specified region.

Parameters

RegionPtr	Specifies the region.
X, *Y*	Specifies the x and y coordinates which define the upper-left corner of the rectangle.
Width, *Height*	Specifies the width and height of the rectangle.

Return Values

RectangleIn	The rectangle is entirely in the specified region.
RectangleOut	The rectangle is entirely out of the specified region.
RectanglePart	The rectangle is partially in the specified region.

Error Code

BadImplementation

Implementation Specifics

This Xlib subroutine is part of AIXwindows Run Time Environment in AIXwindows Environment/6000.

XRefreshKeyboardMapping Subroutine

Purpose

Refreshes stored modifier and keymap information.

Libraries

Enhanced X-Windows Library (**libX11.a**)

FORTRAN 77 Library (**libXfx.a**)

C Syntax

XRefreshKeyboardMapping(*EventMap*)
XMappingEvent **EventMap*;

FORTRAN Syntax

external fxrefreshkeyboardmapping
integer*4 *EventMap*
call fxrefreshkeyboardmapping(*EventMap***)**

Description

The **XRefreshKeyboardMapping** subroutine refreshes the stored modifier and keymap information. Usually, this subroutine is called when a **MappingNotify** event occurs to update client knowledge of the keyboard.

Parameter

EventMap	Specifies the mapping event to be used.

Error Code

BadImplementation

Implementation Specifics

This Xlib subroutine is part of AIXwindows Run Time Environment in AIXwindows Environment/6000.

XRemoveFromSaveSet Subroutine

Purpose

Removes a window from the client's save-set.

Libraries

Enhanced X-Windows Library (**libX11.a**)

FORTRAN 77 Library (**libXfx.a**)

C Syntax

XRemoveFromSaveSet(*DisplayPtr, WindowRemove***)**
**Display ****DisplayPtr*;
Window *WindowRemove*;

FORTRAN Syntax

external fxremovefromsaveset
integer*4 *DisplayPtr*
integer*4 *WindowRemove*
call fxremovefromsaveset(*DisplayPtr, WindowRemove***)**

Description

The **XRemoveFromSaveSet** subroutine removes the specified window and the children of the specified window from the client 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.

Parameters

DisplayPtr	Specifies the connection to the X Server.
WindowRemove	Specifies the window ID of the window to be removed.

Error Codes

BadImplementation

BadMatch

BadWindow

Implementation Specifics

This Xlib subroutine is part of AIXwindows Run Time Environment in AIXwindows Environment/6000.

Related Information

The **ChangeSaveSet** protocol request.

XRemoveHost Subroutine

Purpose

Removes the specified host from the access control list for a display.

Libraries

Enhanced X-Windows Library (**libX11.a**)

FORTRAN 77 Library (**libXfx.a**)

C Syntax

XRemoveHost(*DisplayPtr, Host***)**
Display ***DisplayPtr;**
XHostAddress ***Host;**

FORTRAN Syntax

external fxremovehost
integer*4 *DisplayPtr, Host*
call fxremovehost(*DisplayPtr, Host***)**

Description

The **XRemoveHost** subroutine removes the specified host from the access control list for that display. The server must be on the same host as the client process.

A system removed from the access list can no longer connect to that server. Reset the server, to regain access.

Parameters

DisplayPtr	Specifies the connection to the X Server.
Host	Specifies the network address of the host system.

Error Codes

BadAlloc

BadImplementation

BadValue

Implementation Specifics

This Xlib subroutine is part of AIXwindows Run Time Environment in AIXwindows Environment/6000.

Related Information

The **ChangeHosts** protocol request.

XRemoveHosts Subroutine

Purpose

Removes each specified host.

Libraries

Enhanced X-Windows Library (**libX11.a**)

FORTRAN 77 Library (**libXfx.a**)

C Syntax

XRemoveHosts(*DisplayPtr*, *Hosts*, *NumberHosts*)
Display **DisplayPtr*;
XHostAddress **Hosts*;
int *NumberHosts*;

FORTRAN Syntax

external fxremovehosts
integer*4 *DisplayPtr*, *Hosts*, *NumberHosts*
call fxremovehosts(*DisplayPtr*, *Hosts*, *NumberHosts*)

Description

The **XRemoveHosts** subroutine removes each specified host from the access control list for that display. The display must be on the same host as the client process.

A system removed from the access list cannot connect to that server. You must reset the server to regain access.

Parameters

DisplayPtr	Specifies the connection to the X Server.
Hosts	Specifies the list of hosts to be removed.
NumberHosts	Specifies the number of hosts in the list.

Error Codes

BadAccess

BadImplementation

BadValue.

Implementation Specifics

This Xlib subroutine is part of AIXwindows Run Time Environment in AIXwindows Environment/6000.

Related Information

The **XHostAddress** data structure.

The **ChangeHosts** protocol request.

XReparentWindow Subroutine

Purpose

Changes the parent of a window.

Libraries

Enhanced X-Windows Library (**libX11.a**)

FORTRAN 77 Library (**libXfx.a**)

C Syntax

XReparentWindow(*DisplayPtr*, *WindowID*, *Parent*, *X*, *Y*)
Display **DisplayPtr*;
Window *WindowID*;
Window *Parent*;
int *X*, *Y*;

FORTRAN Syntax

external fxreparentwindow
integer*4 *DisplayPtr*
integer*4 *WindowID*
integer*4 *Parent*
integer*4 *X*, *Y*
call fxreparentwindow(*DisplayPtr*, *WindowID*, *Parent*, *X*, *Y*)

Description

The **XReparentWindow** subroutine reparents the specified window by inserting it as the child of the specified parent. If the specified window is mapped, the **XReparentWindow** subroutine automatically performs the **XUnmapWindow** subroutine on it. Then, the **XReparentWindow** subroutine 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. The *override_redirect* field of the structure returned by this event can be set to the value of **True** or **False**. If the value is **True**, window manager clients normally ignore this event.

Finally, if the specified window was mapped originally, the **XReparentWindow** subroutine performs the **XMapWindow** subroutine on it automatically.

The X Server performs normal exposure processing on formerly obscured windows. The X Server might not generate the **Expose** 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 parent window.
- The new parent window is the specified window or an inferior of the specified window.
- The specified window has a **ParentRelative** background and the new parent window is not the same depth as the specified window.

Parameters

DisplayPtr	Specifies the connection to the X Server.
WindowID	Specifies the window ID.
Parent	Specifies the parent window ID.
X	Specifies the x coordinate, which defines the position of the specified window in the new parent window.
Y	Specifies the y coordinate, which defines the position of the specified window in the new parent window.

Error Codes

BadMatch

BadImplementation

BadWindow

Implementation Specifics

This Xlib subroutine is part of AIXwindows Run Time Environment in AIXwindows Environment/6000.

Related Information

The **XMapWindow** subroutine, **XUnmapWindow** subroutine.

The **ReparentWindow** protocol request.

XResetScreenSaver Subroutine

Purpose

Resets the screen saver.

Libraries

Enhanced X-Windows Library (**libX11.a**)

FORTRAN 77 Library (**libXfx.a**)

C Syntax

XResetScreenSaver(*DisplayPtr***)**
Display ****DisplayPtr*;**

FORTRAN Syntax

external fxresetscreensaver
integer*4 *DisplayPtr*
call fxresetscreensaver(*DisplayPtr***)**

Description

The **XResetScreenSaver** subroutine resets the screen saver.

Parameter

DisplayPtr Specifies the connection to the X Server.

Implementation Specifics

This Xlib subroutine is part of AIXwindows Run Time Environment in AIXwindows Environment/6000.

Related Information

The **XSetScreenSaver** subroutine.

The **ForceScreenSaver** protocol request, **CreateColormap** protocol request.

XResizeWindow Subroutine

Purpose

Changes the size of the specified window.

Libraries

Enhanced X-Windows Library (**libX11.a**)

FORTRAN 77 Library (**libXfx.a**)

C Syntax

XResizeWindow(*DisplayPtr*, *WindowID*, *Width*, *Height*)
Display **DisplayPtr*;
Window *WindowID*;
unsigned int *Width*, *Height*;

FORTRAN Syntax

external fxresizewindow
integer*4 *DisplayPtr*
integer*4 *WindowID*
integer*4 *Width*
integer*4 *Height*
call fxresizewindow(*DisplayPtr, WindowID, Width, Height*)

Description

The **XResizeWindow** subroutine changes the inside dimensions of the specified window. This subroutine 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.

If the *override_redirect* field of the window is the **False** value and another client has selected the **SubstructureRedirectMask** event mask on the parent window, a **ConfigureRequest** event is generated and no further processing is performed.

Parameters

DisplayPtr	Specifies the connection to the X Server.
WindowID	Specifies the window ID.
Width, *Height*	Specifies the width and height of the resized window.

Error Codes

BadImplementation

BadWindow

Implementation Specifics

This Xlib subroutine is part of AIXwindows Run Time Environment in AIXwindows Environment/6000.

Related Information

The **ConfigureWindow** protocol request.

XResourceManagerString Subroutine

Purpose

Returns the resource manager string from the screen of the root window of screen zero.

Libraries

Enhanced X-Windows Library (**libX11.a**)

FORTRAN 77 Library (**libXfx.a**)

C Syntax

char *XResourceManagerString(*DisplayPtr***)**
**Display ****DisplayPtr*;

FORTRAN Syntax

integer*4 fxresourcemanagerstring
external fxresourcemanagerstring
integer*4 *DisplayPtr*
integer*4 *String*
String = **fxresourcemanagerstring(***DisplayPtr***)**

Description

The **XResourceManagerString** subroutine returns the **RESOURCE_MANAGER** property from screen zero of the server's root window.

Parameter

DisplayPtr Specifies the connection to the X Server.

Error Code

BadImplementation

Implementation Specifics

This Xlib subroutine is part of AIXwindows Run Time Environment in AIXwindows Environment/6000.

Related Information

The **XOpenDisplay** subroutine.

XRestackWindows Subroutine

Purpose

Restacks a set of windows from top to bottom.

Libraries

Enhanced X-Windows Library (**libX11.a**)

FORTRAN 77 Library (**libXfx.a**)

C Syntax

XRestackWindows(*DisplayPtr*, *Windows*, *NumberWindows*)
Display **DisplayPtr*;
Window *Windows*[];
int *NumberWindows*;

FORTRAN Syntax

external fxrestackwindows
integer*4 *DisplayPtr*
integer*4 *Windows*
integer*4 *NumberWindows*
call fxrestackwindows(*DisplayPtr*, *Windows*, *NumberWindows*)

Description

The **XRestackWindows** subroutine restacks the windows from top to bottom in the specified order. The stacking order of the first window in the windows array is not affected, but the other windows in the array are stacked underneath the first window in the specified order. For each window in the windows array that is not a child of the specified window, a **BadMatch** error results.

If the *override_redirect* field of the window is the **False** value and another client has selected the **SubstructureRedirectMask** event mask 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 are restacked from top to bottom.

Parameters

DisplayPtr	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 window.
NumberWindows	Specifies the number of windows to be restacked.

Error Codes

BadWindow

BadImplementation

Implementation Specifics

This Xlib subroutine is part of AIXwindows Run Time Environment in AIXwindows Environment/6000.

Related Information

The **ConfigureWindow** protocol request.

XRotateBuffers Subroutine

Purpose

Rotates the cut buffers.

Libraries

Enhanced X-Windows Library (**libX11.a**)

FORTRAN 77 Library (**libXfx.a**)

C Syntax

XRotateBuffers(*DisplayPtr*, *Rotate***)**
Display **DisplayPtr*;
int *Rotate*;

FORTRAN Syntax

external fxrotatebuffers
integer*4 *DisplayPtr*, *Rotate*
call fxrotatebuffers(*DisplayPtr*, *Rotate***)**

Description

The **XRotateBuffers** subroutine rotates all eight cut buffers. Buffer `0` becomes buffer `n`; buffer `1` becomes `n+1 mod 8`. The current cut buffer numbering is global to the display. If any of the eight buffers has not been created, the **XRotateBuffers** subroutine generates an error.

Parameters

DisplayPtr	Specifies the connection to the X Server.
Rotate	Specifies the amount of rotation.

Error Codes

BadImplementation

BadMatch

Implementation Specifics

This Xlib subroutine is part of AIXwindows Run Time Environment in AIXwindows Environment/6000.

Related Information

The **RotateProperties** protocol request.

XRotateWindowProperties Subroutine

Purpose

Rotates the property list of a window.

Libraries

Enhanced X-Windows Library (**libX11.a**)

FORTRAN 77 Library (**libXfx.a**)

C Syntax

XRotateWindowProperties(*DisplayPtr*, *WindowID*, *Properties*, *NumberProperties*, *NumberPositions*);
Display **DisplayPtr*;
Window *WindowID*;
Atom *Properties*[];
int *NumberProperties*;
int *NumberPositions*;

FORTRAN Syntax

external fxrotatewindowproperties
integer*4 *DisplayPtr*
integer*4 *WindowID*
integer*4 *Properties*
integer*4 *NumberProperties*, *NumberPositions*
call fxrotatewindowproperties(*DisplayPtr*, *WindowID*, *Properties*, *NumberProperties*, *NumberPositions*)

Description

The **XRotateWindowProperties** subroutine rotates properties in the properties list. It rotates the integer values of the *NumberPositions* parameter around a virtual ring of property names (right for positive values, left for negative values).

The **XRotateWindowProperties** subroutine rotates the properties of a window. The number of properties is specified in the *NumberProperties* parameter. Where the *NumberProperties* parameter represented by `I`, and the *NumberPositions* parameter represented by `n`, the **XRotateWindowProperties** subroutine causes each property to become `n+1 mod I`.

If the value of the *NumberPositions* parameter is a nonzero value, the X Server generates a **PropertyNotify** event for each property in the order listed in the array.

An error is generated and no properties are changed if an atom occurs more than once in the list, if a property name is undefined, or if it does not exist.

Parameters

DisplayPtr	Specifies the connection to the X Server.
WindowID	Specifies the window ID.
Properties	Specifies the array of properties to be rotated.
NumberProperties	Specifies the length of the properties array.
NumberPositions	Specifies the amount of rotation.

Error Codes

BadAtom

BadImplementation

BadMatch

BadWindow

Implementation Specifics

This Xlib subroutine is part of AIXwindows Run Time Environment in AIXwindows Environment/6000.

Related Information

The **RotateProperties** protocol request.

XrmGetFileDatabase Subroutine

Purpose

Retrieves a database from disk.

Libraries

Enhanced X-Windows Library (**libX11.a**)

FORTRAN 77 Library (**libXfx.a**)

C Syntax

XrmDatabase XrmGetFileDatabase(*DatabaseFilename***)**
char ****DatabaseFilename*;**

FORTRAN Syntax

integer*4 fxrmgetfiledatabase
external fxrmgetfiledatabase
character*256 *DatabaseFilename*
integer*4 *ResourceDatabase*
ResourceDatabase **= fxrmgetfiledatabase(***DatabaseFilename***)**

Description

The **XrmGetFileDatabase** subroutine retrieves a database from disk. This subroutine opens the specified file, creates a new resource database, and loads it with the specifications read from the specified file. The specified file must contain lines in the format accepted by the **XrmPutLineResource** subroutine. A value of the **XrmDatabase** type is returned.

Parameter

DatabaseFilename	Specifies the resource database file name.

Return Values

NULL	The subroutine cannot open the specified file. Otherwise, it returns a value of type **XrmDatabase**.

Error Code

BadImplementation

Implementation Specifics

This Xlib subroutine is part of AIXwindows Run Time Environment in AIXwindows Environment/6000.

Related Information

The **XrmPutLineResource** subroutine.

XrmGetResource Subroutine

Purpose

Retrieves a resource from a database.

Libraries

Enhanced X-Windows Library (**libX.a**)

FORTRAN 77 Library (**libXfx.a**)

C Syntax

Bool XrmGetResource(*Database*, *StringName*, *StringClass*, *StringTypeReturn*, *StringValueReturn*)
XrmDatabase *Database*;
char **StringName*;
char **StringClass*;
char ***StringTypeReturn*;
XrmValue **StringValueReturn*;

FORTRAN Syntax

integer*4 fxrmgetresource
external fxrmgetresource
integer*4 *Database*
character*256 *StringName*
character*256 *StringClass*
integer*4 *StringTypeReturn*
integer*4 *StringValueReturn*
integer*4 *Bool*
Bool = **fxrmgetresource**(*Database*, *StringName*, *StringClass*, *StringTypeReturn*, *StringValueReturn*)

Description

The **XrmGetResource** subroutine retrieves a resource from a database.

Parameters

Database	Specifies the database to be used.
StringName	Specifies the fully qualified name (as a string) of the value being retrieved.
StringClass	Specifies the fully qualified class (as a string) of the value being retrieved.
StringTypeReturn	Returns a pointer (as a string) to the representation type of the destination.
StringValueReturn	Returns the value in the database.

Implementation Specifics

This Xlib subroutine is part of AIXwindows Run Time Environment in AIXwindows Environment/6000.

XrmGetStringDatabase Subroutine

Purpose

Creates a database from a specified string.

Libraries

Enhanced X-Windows Library (**libX11.a**)

FORTRAN 77 Library(**libXfx.a**)

C Syntax

XrmDatabase XrmGetStringDatabase(*Database***)**
char ****Database*;**

FORTRAN Syntax

integer*4 fxrmgetstringdatabase
external fxrmgetstringdatabase
character*256 *Database*
integer*4 *ResourceDatabase*
ResourceDatabase = **fxrmgetstringdatabase(***Database***)**

Description

The **XrmGetStringDatabase** subroutine creates a new database and stores the resources specified in the specified null-terminated string. This subroutine reads the information out of a string. Each line is separated by a new-line character in the format accepted by the **XrmPutLineResource** subroutine.

Parameter

Database Specifies the database contents using a string.

Error Code

BadImplementation

Implementation Specifics

This Xlib subroutine is part of AIXwindows Run Time Environment in AIXwindows Environment/6000.

Related Information

The **XrmPutLineResource** subroutine.

XrmInitialize Subroutine

Purpose

Initializes the resource manager.

Libraries

Enhanced X-Windows Library (**libX11.a**)

FORTRAN 77 Library(**libXfx.a**)

C Syntax

void XrmInitialize( )

FORTRAN Syntax

external fxrminitialize
call fxrminitialize( )

Description

The **XrmInititalize** subroutine initializes the resource manager.

Error Code

BadImplementation

Implementation Specifics

This Xlib subroutine is part of AIXwindows Run Time Environment in AIXwindows Environment/6000.

Related Information

Basic Resource Manager Definitions

XrmMergeDatabases Subroutine

Purpose

Merges two databases.

Libraries

Enhanced X-Windows Library (**libX11.a**)

FORTRAN 77 Library (**libXfx.a**)

C Syntax

void XrmMergeDatabases(*SourceDatabase*, *TargetDatabase***)**
XrmDatabase *SourceDatabase*, ******TargetDatabase*;

FORTRAN Syntax

external fxrmmergedatabases
integer*4 *SourceDatabase*
integer*4 *TargetDatabase*
call fxrmmergedatabases(*SourceDatabase*, *TargetDatabase***)**

Description

The **XrmMergeDatabases** subroutine merges the contents of one database into another. This subroutine may overwrite entries in the destination database. The **XrmMergeDatabases** subroutine is used to combine databases such as an application specific database of defaults with a database of user preferences. The original database is destroyed.

Parameters

SourceDatabase	Specifies the resource database to be merged into the target database.
TargetDatabase	Specifies a pointer to the resource database where the source database is to be merged.

Error Code

BadImplementation

Implementation Specifics

This Xlib subroutine is part of AIXwindows Run Time Environment in AIXwindows Environment/6000.

XrmParseCommand Subroutine

Purpose

Stores command options into a database.

Libraries

Enhanced X-Windows Library (**libX11.a**)

FORTRAN 77 Library (**libXfx.a**)

C Syntax

void XrmParseCommand(*Database*, *Table*, *TableCount*, *Name*,*argcInOut*, *argvInOut*)
XrmDatabase **Database*;
XrmOptionDescList *Table*;
int *TableCount*;
char **Name*;
int **argcInOut*;
char ***argvInOut*;

FORTRAN Syntax

external fxrmparsecommand
integer*4 *Database*, *Table*, *TableCount*
character*256 *Name*
integer*4 *argcInOut*, *argvInOut*
call fxrmparsecommand(*Database*, *Table*, *TableCount*, *Name*, *argcInOut*, *argvInOut*)

Description

The **XrmParseCommand** subroutine loads a database from a C language command line according to the following:

- It parses an *argc, argv* parameter pair according to the specified option table.
- It loads recognized options into the specified database with type "String".
- It modifies the *argc, argv* parameter pair to remove all recognized options.

If the resource database is **NULL**, a new resource database is created and a pointer is returned to the new resource database.

The specified table is used to parse the command line. Recognized entries in the table are removed from the *argvInOut* parameter, 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 the **XrmOptionNoArg** subroutine.

Use the application name as the *Name* parameter. This parameter is prefixed to the *ResourceName* parameter in the option table before storing the specification.

The *argcInOut* parameter specifies the number of arguments in the *argvInOut* parameter and is set to the number of arguments that remain.

Parameters

Database	Specifies a pointer to the resource database.
Table	Specifies the table of command line arguments to be parsed.
TableCount	Specifies the number of entries in the table.
Name	Specifies the application name.
argcInOut	Specifies the number of arguments before the call. Returns the number of arguments remaining after the call.
argvInOut	Specifies a pointer to the command line arguments before the call. Returns the matched arguments removed after the call.

Error Code

BadImplementation

Implementation Specifics

This Xlib subroutine is part of AIXwindows Run Time Environment in AIXwindows Environment/6000.

Related Information

The **XrmOptionDescRec** data structure.

XrmPutFileDatabase Subroutine

Purpose

Copies a database into a specified file.

Libraries

Enhanced X-Windows Library (**libX11.a**)

FORTRAN 77 Library (**libXfx.a**)

C Syntax

void XrmPutFileDatabase(*Database*, *Filename***)**
XrmDatabase *Database*;
char **Filename*;

FORTRAN Syntax

external fxrmputfiledatabase
integer*4 *Database*
character*256 *Filename*
call fxrmputfiledatabase(*Database*, *Filename***)**

Description

The **XrmPutFileDatabase** subroutine stores a copy of the current database of an application in the specified file. The file is an ASCII text file that contains lines in the format accepted by the **XrmPutLineResource** subroutine.

Parameters

Database	Specifies the database to be used.
Filename	Specifies the file name for the stored database.

Error Code

BadImplementation

Implementation Specifics

This Xlib subroutine is part of AIXwindows Run Time Environment in AIXwindows Environment/6000.

Related Information

The **XrmPutLineResource** subroutine.

XrmPutLineResource Subroutine

Purpose

Stores a single resource entry into a database.

Libraries

Enhanced X-Windows Library (**libX11.a**)

FORTRAN 77 Library (**libXfx.a**)

C Syntax

void XrmPutLineResource(*Database*, *Line***)**
XrmDatabase **Database*;
char **Line*;

FORTRAN Syntax

external fxrmputlineresource
integer*4 *Database*
character*256 *Line*
call fxrmputlineresource(*Database*, *Line***)**

Description

The **XrmPutLineResource** subroutine 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 white space before or after the name or a colon in the *Line* parameter is ignored. The value is ended by a new-line character or a **NULL** character.

The value pair can contain embedded new-line characters prefixed by the \n (backslash n) character pair, which are removed before the value is stored in the database.

For example, the *Line* parameter might have the value:

```
aixterm*background:green\n
```

Null-terminated strings without a new-line character are also permitted.

If the *Database* parameter is **NULL**, a new resource database is created and a pointer to the new resource database is returned in the *Database* parameter.

Parameters

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.

Error Code

BadImplementation

Implementation Specifics

This Xlib subroutine is part of AIXwindows Run Time Environment in AIXwindows Environment/6000.

XrmPutResource Subroutine

Purpose

Stores a resource into a database.

Libraries

Enhanced X-Windows Library (**libX11.a**)

FORTRAN 77 Library (**libXfx.a**)

C Syntax

void XrmPutResource(*Database*, *Specifier*, *Type*,
Value)
XrmDatabase **Database*;
char **Specifier*;
char **Type*;
XrmValue **Value*;

FORTRAN Syntax

external fxrmputresource
integer*4 *Database*
character*256 *Specifier*
character*256 *Type*
integer*4 *Value*
call fxrmputresource(*Database*, *Specifier*, *Type*, *Value*)

Description

The **XrmPutResource** subroutine calls the **XrmStringToBindingQuarkList** subroutine, followed by the following expression:

```
XrmQPutResource(database, bindings, quarks,
   XrmStringToQuarkList(type), value)
```

The **XrmPutResource** subroutine stores resources into the database. This subroutine takes a partial resource specification, a representation type, and a value. This value is copied into the specified database.

If the *Database* parameter is **NULL**, a new resource database is created and a pointer to the new resource database is returned in this parameter.

Parameters

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.

Error Code

BadImplementation

Implementation Specifics

This Xlib subroutine is part of AIXwindows Run Time Environment in AIXwindows Environment/6000.

Related Information

The **XrmStringToBindingQuarkList** subroutine.

XrmPutStringResource Subroutine

Purpose

Stores a string resource into a database.

Libraries

Enhanced X-Windows Library (**libX11.a**)

FORTRAN 77 Library (**libXfx.a**)

C Syntax

void XrmPutStringResource(*Database*, *Resource*, *Value***)**
XrmDatabase **Database*;
char **Resource*;
char **Value*;

FORTRAN Syntax

external fxrmputstringresource
integer*4 *Database*
character*256 *Resource*
character*256 *Value*
call fxrmputstringresource(*Database*, *Resource*, *Value***)**

Description

The **XrmPutStringResource** subroutine adds a resource with the specified value to the specified database. This subroutine takes both the resource and the value as strings, converts them to quarks, and then calls the **XrmQPutResource** subroutine.

If the *Database* parameter is the **NULL** value, a new resource database is created and a pointer to the new database is returned.

Parameters

Database	Specifies a pointer to the resource database.
Resource	Specifies the resource as a string.
Value	Specifies the value of the resource.

Error Code

BadImplementation

Implementation Specifics

This Xlib subroutine is part of AIXwindows Run Time Environment in AIXwindows Environment/6000.

Related Information

The **XrmQPutResource** subroutine.

XrmQGetResource Subroutine

Purpose

Retrieves a resource from a database.

Libraries

Enhanced X-Windows Library (**libX11.a**)

FORTRAN 77 Library (**libXfx.a**)

C Syntax

Boolean XrmQGetResource(*Database*, *QuarkName*, *QuarkClass*, *QuarkTypeReturn*, *ValueReturn*)
XrmDatabase *Database*;
XrmNameList *QuarkName*;
XrmClassList *QuarkClass*;
XrmRepresentation **QuarkTypeReturn*;
XrmValue **ValueReturn*;

FORTRAN Syntax

integer*4 fxrmqgetresource
external fxrmqgetresource
integer*4 *Database*
integer*4 *QuarkName*
integer*4 *QuarkClass*
integer*4 *QuarkTypeReturn*
integer*4 *ValueReturn*
integer*4 *Boolean*
Boolean = **fxrmqgetresource**(*Database*, *QuarkName*, *QuarkClass*, *QuarkTypeReturn*, *ValueReturn*)

Description

The **XrmQGetResource** subroutine gets a resource from a resource database. It takes a fully qualified name-class pair, a destination resource representation, and the address of a value (size, address pair). The value and returned type point into database memory; therefore, you must not modify the data.

The database only frees or overwrites entries on the **XrmPutResourc**, **XrmQPutResource**, or **XrmMergeDatabases** subroutine. A client that is not storing new values into the database or is not merging the database should be safe using the address passed back at any time until it exists.

Parameters

Database	Specifies the database to be used.
QuarkName	Specifies the fully qualified name, as a quark, of the value being retrieved.
QuarkClass	Specifies the fully qualified class, as a quark, of the value being retrieved.
QuarkTypeReturn	Returns a pointer, as a quark, to the representation type of the destination.
ValueReturn	Returns the value in the database.

Return Values

False	The resource does not exist.
True	The resource exists.

Error Code

BadImplementation

Implementation Specifics

This Xlib subroutine is part of AIXwindows Run Time Environment in AIXwindows Environment/6000.

XrmQGetSearchList Subroutine

Purpose

Gets a list of database levels.

Libraries

Enhanced X-Windows Library (**libX11.a**)

FORTRAN 77 Library (**libXfx.a**)

C Syntax

Bool XrmQGetSearchList(*Database*, *Names*, *Classes*, *ListReturn*, *ListLength***)**
XrmDatabase *Database*;
XrmNameList *Names*;
XrmClassList *Classes*;
XrmSearchList *ListReturn*;
int *ListLength*;

FORTRAN Syntax

integer*4 fxrmqgetsearchlist
external fxrmqgetsearchlist
integer*4 *Database*
integer*4 *Names*
integer*4 *Classes*
integer*4 *ListReturn*
integer*4 *ListLength*
integer*4 *Bool*
Bool = **fxrmqgetsearchlist(***Database*, *Names*, *Classes*, *ListReturn*, *ListLength***)**

Description

The **XrmQGetSearchList** subroutine takes a list of names and classes and returns a list of database levels where a match might occur. The returned list is in best-to-worst order. It uses the same algorithm as the **XrmGetResource** subroutine for determining precedence.

Sufficient space must be allocated before using the **XrmQGetSearchList** subroutine. The requirement for allocation size is dependent upon the number of levels and pattern-matching characters 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.

When using the **XrmQGetSearchList** subroutine 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.

Parameters

Database	Specifies the database to be used.
Names	Specifies a list of resource names.
Classes	Specifies a list of resource classes.
ListReturn	Returns a search list for further use. Sufficient space must be allocated before using the **XrmQGetSearchList** subroutine.
ListLength	Specifies the number of entries (not the byte size) allocated for the *ListReturn* parameter.

Return Values

True	Indicates that sufficient space is allocated for the search list.
False	Indicates that insufficient space is allocated for the search list.

Error Code

BadImplementation

Implementation Specifics

This Xlib subroutine is part of AIXwindows Run Time Environment in AIXwindows Environment/6000.

XrmQGetSearchResource Subroutine

Purpose

Searches resource database levels for a specified resource.

Libraries

Enhanced X-Windows Library (**libX11.a**)

FORTRAN 77 Library (**libXfx.a**)

C Syntax

Bool XrmQGetSearchResource(*List*, *Name*, *Class*, *TypeReturn*,*ValueReturn*)
XrmSearchList *List*;
XrmName *Name*;
XrmClass *Class*;
XrmRepresentation **TypeReturn*;
XrmValue **ValueReturn*;

FORTRAN Syntax

integer*4 fxrmqgetsearchresource
external fxrmqgetsearchresource
integer*4 *List, Name*
integer*4 *Class*, *TypeReturn*, *ValueReturn*
integer*4 *Bool*
Bool = **fxrmqgetsearchresource**(*List*, *Name*, *Class*, *TypeReturn*, *ValueReturn*)

Description

The **XrmQGetSearchResource** subroutine searches the specified database levels for the resource identified by the specified name and class. The search stops with the first match.

If a call to the **XrmQGetSearchList** subroutine with a name and class list containing all but the last component of a resource name is followed by a call to the **XrmQGetSearchResource** subroutine with the last component name and class, the same database entry is returned as when a call is made to either the **XrmGetResource** or **XrmQGetResource** subroutine with the fully qualified name and class list.

Parameters

List	Specifies the search list returned by the **XrmQGetSearchList** subroutine.
Name	Specifies the resource name.
Class	Specifies the resource class.
TypeReturn	Returns data representation type.
ValueReturn	Returns the value in the database.

Return Values

True	Indicates that the requested resource is found.
False	Indicates that the requested resource is not found.

Error Code

BadImplementation

Implementation Specifics

This Xlib subroutine is part of AIXwindows Run Time Environment in AIXwindows Environment/6000.

XrmQPutResource Subroutine

Purpose

Stores resources into a database.

Libraries

Enhanced X-Windows Library (**libX11.a**)

FORTRAN 77 Library (**libXfx.a**)

C Syntax

void XrmQPutResource(*Database*, *Bindings*, *Quarks*, *Type*, *Value***);**
XrmDatabase **Database*;
XrmBindingList *Bindings*;
XrmQuarkList *Quarks*;
XrmRepresentation *Type*;
XrmValue **Value*;

FORTRAN Syntax

external fxrmqputresource
integer*4 *Database*
integer*4 *Bindings*
integer*4 *Quarks*
integer*4 *Type*
integer*4 *Value*
call fxrmqputresource(*Database*, *Bindings*, *Quarks*, *Type*, *Value***)**

Description

The **XrmQPutResource** subroutine stores resources into a database. It takes a partial resource specification, a representation type, and a value. This value is copied into the specified database.

If the *Database* parameter is a value of **NULL**, the **XrmQPutResource** subroutine creates a new resource database and returns a pointer to it.

Parameters

Database	Specifies a pointer to a resource database.
Bindings	Specifies a list of bindings.
Quarks	Specifies the complete or partial name, or class list, of the resource to be stored.
Type	Specifies the type of the resource.
Value	Specifies, as a string, the value of the resource.

Error Code

BadImplementation

Implementation Specifics

This Xlib subroutine is part of AIXwindows Run Time Environment in AIXwindows Environment/6000.

XrmQPutStringResource Subroutine

Purpose

Stores a string resource into a database using quarks as a specification.

Libraries

Enhanced X-Windows Library (**libX11.a**)

FORTRAN 77 Library (**libXfx.a**)

C Syntax

void XrmQPutStringResource(*Database*, *Bindings*, *Quarks*, *Value*)
XrmDatabase **Database*;
XrmBindingList *Bindings*;
XrmQuarkList *Quarks*;
char **Value*;

FORTRAN Syntax

external fxrmqputstringresource
integer*4 *Database*
integer*4 *Bindings*
integer*4 *Quarks*
character*256 *Value*
call fxrmqputstringresource(*Database*, *Bindings*, *Quark*, *Value*)

Description

The **XrmQPutStringResource** subroutine stores a string resource into a database using quarks as a specification. It constructs an **XrmValue** data structure for the value string by calling the **strlen** function, which computes the size. It then calls the **XrmQPutResource** subroutine, using a string data type.

If the *Database* parameter is **NULL**, the **XrmQPutStringResource** subroutine creates a new resource database and returns a pointer to it.

Parameters

Database	Specifies a pointer to the resource database.
Bindings	Specifies a list of bindings.
Quarks	Specifies the complete or partial name, or class list, of the resource.
Value	Specifies as a string the value of the resource.

Error Code

BadImplementation

Implementation Specifics

This Xlib subroutine is part of AIXwindows Run Time Environment in AIXwindows Environment/6000.

XrmQuarkToString Subroutine

Purpose

Converts a quark to a character string.

Libraries

Enhanced X-Windows Library (**libX11.a**)

FORTRAN 77 Library (**libXfx.a**)

C Syntax

#define XrmNameToString(*Name***) XrmQuarkToString(***Name***)**
#define XrmClassToString(*Class***) XrmQuarkToString(***Class***)**
#define XrmRepresentationToString(*Type***) XrmQuarkToString(***Type***)**

char *XrmQuarkToString(*Quark***)**
XrmQuark *Quark*;

FORTRAN Syntax

character*256 fxrmquarktostring
external fxrmquarktostring
integer*4 *Quark*
character*256 *QuarkToString*
QuarkToString = **fxrmquarktostring(***Quark***)**

Description

The **XrmQuarkToString** subroutine converts a quark to a string. The string pointed to by the return value must not be modified or freed.

Parameter

Quark	Specifies the quark to be converted to a string.

Return Values

NULL	Indicates that no string exists for the specified quark.
String	String converted from *Quark*.

Error Code

BadImplementation

Implementation Specifics

This Xlib subroutine is part of AIXwindows Run Time Environment in AIXwindows Environment/6000.

XrmStringToBindingQuarkList Subroutine

Purpose

Converts a string having one or more components to a binding list and to a quark list.

Libraries

Enhanced X-Windows Library (**libX11.a**)

FORTRAN 77 Library (**libXfx.a**)

C Syntax

XrmStringToBindingQuarkList(*String*, *BindingsReturn*, *QuarksReturn*)
char **String*;
XrmBindingList *BindingsReturn*;
XrmQuarkList *QuarksReturn*;

FORTRAN Syntax

external fxrmstringtobindingquarklist
character*256 *String*;
integer*4 *BindingsReturn*;
integer*4 *QuarksReturn*

call fxrmstringtobindingquarklist(*String*, *BindingsReturn*, *QuarksReturn*)

Description

The **XrmStringToBindingQuarkList** subroutine converts a string having one or more components to a binding list and to a quark list.

Component names in the list are separated by the . (period) or the * (asterisk) character. If the first character of the string is not a period or an asterisk, a period is inserted by default.

For example, `*a.b*c` becomes the following:

`Quarks`	`a`	`b`	`c`
`Bindings`	`loose`	`tight`	`loose`

Sufficient space must be allocated for binding and quarks lists before calling the **XrmStringToBindingQuarkList** subroutine.

Parameters

String	Specifies the string for which a quark is to be allocated.
BindingsReturn	Returns the binding list.
QuarksReturn	Returns the quark list.

Error Code

BadImplementation

Implementation Specifics

This Xlib subroutine is part of AIXwindows Run Time Environment in AIXwindows Environment/6000.

XrmStringToQuark Subroutine

Purpose

Converts a character string to a quark.

Libraries

Enhanced X-Windows Library (**libX11.a**)

FORTRAN 77 Library (**libXfx.a**)

C Syntax

#define XrmStringToName(*String***) XrmStringToQuark(***String***)**
#define XrmStringToClass(*String***) XrmStringToQuark(***String***)**
#define XrmStringToRepresentation(*String***) XrmStringToQuark(***String***)**

XrmQuark XrmStringToQuark(*String***);**
char ****String*;**

FORTRAN Syntax

integer*4 fxrmstringtoquark
external fxrmstringtoquark
character*256 *String*
integer*4 *Quark*
Quark **= fxrmstringtoquark(***String***)**

Description

The **XrmStringToQuark** subroutine converts a string to a quark. The string pointed to by the return value must not be modified or freed.

Parameter

String Specifies the string to be converted to a quark.

Error Code

BadImplementation

Implementation Specifics

This Xlib subroutine is part of AIXwindows Run Time Environment in AIXwindows Environment/6000.

XrmStringToQuarkList Subroutine

Purpose

Converts a character string with one or more components to a quark list.

Libraries

Enhanced X-Windows Library (**libX11.a**)

FORTRAN 77 Library (**libXfx.a**)

C Syntax

#define XrmStringToNameList(*String,Name***) XrmStringToQuarkList((***String***),(***Name***))**
#define XrmStringToClassList(*String,Class***) XrmStringToQuarkList((***String***),(***Class***))**

void XrmStringToQuarkList(*String, QuarksReturn***);**
char ****String*;**
XrmQuarkList *QuarksReturn*;

FORTRAN Syntax

external fxrmstringtoquarklist
character*256 *String*
integer*4 *QuarksReturn*
call fxrmstringtoquarklist(*String, QuarksReturn***)**

Description

The **XrmStringToQuarkList** subroutine converts a null-terminated string to a list of quarks. This string is generally a fully qualified name. The components of the string are separated by the . (period) or * (asterisk) character.

A binding list is a list of the **XrmBindingList** type and indicates if components in name or class lists are bound tightly or loosely. Components are bound loosely when pattern-matching characters are specified.

The **XrmBindTightly** value indicates that a period separates the components. The **XrmBindLoosely** value indicates that an asterisk separates the components.

Parameters

String	Specifies the string to be converted to a quark.
QuarksReturn	Returns the list of quarks.

Error Code

BadImplementation

Implementation Specifics

This Xlib subroutine is part of AIXwindows Run Time Environment in AIXwindows Environment/6000.

XrmUniqueQuark Subroutine

Purpose

Allocates a new quark.

Libraries

Enhanced X-Windows Library (**libX11.a**)

FORTRAN 77 Library (**libXfx.a**)

C Syntax

XrmQuark XrmUniqueQuark( )

FORTRAN Syntax

integer*4 fxrmuniquequark
external fxrmuniquequark
integer*4 *Quark*
Quark **= fxrmuniquequark( )**

Description

The **XrmUniqueQuark** subroutine allocates a new quark and is guaranteed not to represent any string known to the resource manager.

Error Code

BadImplementation

Implementation Specifics

This Xlib subroutine is part of AIXwindows Run Time Environment in AIXwindows Environment/6000.

XSaveContext Subroutine

Purpose

Stores data associated with a window and its context type.

Libraries

Enhanced X-Windows Library (**libX11.a**)

FORTRAN 77 Library (**libXfx.a**)

C Syntax

int XSaveContext(*DisplayPtr*, *WindowID*, *Context*, *Data***);**
Display **DisplayPtr*;
Window *WindowID*;
XContext *Context*;
caddr_t *Data*;

FORTRAN Syntax

integer*4 fxsavecontext
external fxsavecontext
integer*4 *DisplayPtr*, *WindowID*, *Context*
integer*4 *Data*
integer*4 *Status*
Status = **fxsavecontext(***DisplayPtr*, *WindowID*, *Context*, *Data***)**

Description

The **XSaveContext** subroutine saves a data value that corresponds to a window and its context type. If an entry with the specified window and its context type already exists, the **XSaveContext** subroutine overrides that entry with the specified context.

Parameters

DisplayPtr	Specifies the connection to the X Server.
WindowID	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 context type.

Return Values

0	Indicates that the **XSaveContext** subroutine runs successfully.
Nonzero	Indicates that the **XSaveContext** subroutine cannot run.

Error Codes

BadImplementation

XCNOMEM (out of memory)

Implementation Specifics

This Xlib subroutine is part of AIXwindows Run Time Environment in AIXwindows Environment/6000.

XSelectInput Subroutine

Purpose

Selects events to be reported to the client.

Libraries

Enhanced X-Windows Library (**libX11.a**)

FORTRAN 77 Library (**libXfx.a**)

C Syntax

XSelectInput(*DisplayPtr*, *WindowID*, *EventMask*);
Display **DisplayPtr*;
Window *WindowID*;
unsigned long *EventMask*;

FORTRAN Syntax

external fxselectinput
integer*4 *DisplayPtr*
integer*4 *WindowID*
integer*4 *EventMask*
call fxselectinput(*DisplayPtr*, *WindowID*, *EventMask*)

Description

The **XSelectInput** subroutine requests that the X Server report the events associated with a specified event mask. By default, Enhanced X-Windows does not report these events.

Events are reported relative to a window. If a device event is not relevant to the specified window, the window usually propagates to the closest ancestor that is relevant provided that the *do_not_propagate_mask* parameter does not prohibit it.

Setting the *EventMask* parameter of a window overrides any previous call for the same window, but not for 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 their event masks are disjointed. When the X Server generates an event, it reports it to all interested clients.
- Only one client at a time can select the **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 X Server reports the event to all interested clients.

Note: Setting the *event_mask* field of the **XSetWindowAttributes** data structure when using the **XCreateWindow** and **XChangeWindowAttributes** subroutines indicates to the server that it should report events to a client applications.

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 the XGrabPointer subroutine. This ensures that a window will see the **ButtonRelease** event corresponding to the **ButtonPress** event, even though the most may have exited the window in the meantime.

If a client passes **PointerMotionMask**, the X Server sends **MotionNotify** events independent of the estate of the pointer buttons. Instead, the client passes one or more of the event masks **Button1MotionMask** . . . .**Button5MotionMask**, the X Server generates **MotionNotify** events only when one or more of the specified buttons are pressed. These masks are used to request **MotionNotify** events only when particular buttons are held down.

Parameters

DisplayPtr	Specifies the connection to the X Server.
WindowID	Specifies the window ID of the window for which events are to be reported.
EventMask	Specifies the event mask.

Error Codes

BadImplementation

BadWindow

Implementation Specifics

This Xlib subroutine is part of AIXwindows Run Time Environment in AIXwindows Environment/6000.

Related Information

The **ChangeWindowAttributes** protocol request.

XSendEvent Subroutine

Purpose

Sends an event to a specified window.

Libraries

Enhanced X-Windows Library (**libX11.a**)

FORTRAN 77 Library (**libXfx.a**)

C Syntax

Status XSendEvent(*DisplayPtr*, *WindowID*, *Propagate*, *EventMask*, *EventSend*)
Display **DisplayPtr*;
Window *WindowID*;
Bool *Propagate*;
unsigned long *EventMask*;
XEvent **EventSend*;

FORTRAN Syntax

external fxsendevent
integer*4 *DisplayPtr*
integer*4 *WindowID*, *Propagate*, *EventMask*, *EventSend*
call fxsendevent(*DisplayPtr*, *WindowID*, *Propagate*, *EventMask*, *EventSend*)

Description

The **XSendEvent** subroutine identifies a destination window, determines which clients should receive specified events, and ignores any active grabs. This subroutine requires that a valid event mask be specified.

If the *EventMask* parameter is a value of **NULL**, the event is sent to the client that created the destination window. If that client no longer exists, no event is sent.

The event sent by the **XEvent** subroutine must be one of the core events or one of the events defined by an extension so that the X Server can correctly byte-swap the contents.

Parameters

DisplayPtr — Specifies the connection to the X Server.

WindowID — Specifies the window ID of the destination window. The *WindowID* parameter can be either of the following values:

PointerWindow — The destination window is the window that contains the pointer.

InputFocus — If 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.

Propagate Specifies a Boolean value. The *Propagate* parameter specifies which clients receive specified events, and can be either of the following values:

False The event is sent to every client selecting on destination any of the event types in the *EventMask* parameter.

True If no clients have selected on destination any of the event types in the *EventMask* parameter, the destination is replaced with the closest ancestor of destination for which some client has selected a type in the *EventMask* parameter and for which no intervening window has that type as its *DoNotPropagateMask* parameter. If no such window exists or if the window is an ancestor of the focus window and the **InputFocus** value 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 values specified in the *EventMask* parameter.

EventMask Specifies the event mask as the bit-wiseinclusive OR of one or more of the valid event mask bits.

EventSend Specifies a pointer to the event to be sent.

Return Values

0 Indicates that the conversion to wire protocol format does not succeed.

Nonzero Indicates that the conversion to wire protocol format succeeds.

Error Codes

BadImplementation

BadValue

BadWindow

Implementation Specifics

This Xlib subroutine is part of AIXwindows Run Time Environment in AIXwindows Environment/6000.

Related Information

The **SendEvent** protocol request.

XSetAccessControl Subroutine

Purpose

Changes access control.

Libraries

Enhanced X-Windows Library (**libX11.a**)

FORTRAN 77 Library (**libXfx.a**)

C Syntax

XSetAccessControl(*DisplayPtr*, *Mode*);
Display **DisplayPtr*;
int *Mode*;

FORTRAN Syntax

external fxsetaccesscontrol
integer*4 *DisplayPtr*, *Mode*
call fxsetaccesscontrol(*DisplayPtr*, *Mode*)

Description

The **XSetAccessControl** subroutine enables or disables the use of the access control list at each connection setup.

Either 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 the **XSetAccessControl** subroutine to run successfully.

Parameters

DisplayPtr	Specifies the connection to the X Server.
Mode	Specifies the mode as the **EnableAccess** or **DisableAccess** value.

Error Codes

BadAccess

BadImplementation

BadValue

Implementation Specifics

This Xlib subroutine is part of AIXwindows Run Time Environment in AIXwindows Environment/6000.

Related Information

The **SetAccessControl** protocol request.

XSetAfterFunction Subroutine

Purpose

Sets the subroutine to be called after another function generates a protocol request.

Libraries

Enhanced X-Windows Library (**libX11.a**)

FORTRAN 77 Library (**libXfx.a**)

C Syntax

int (*XSetAfterFunction(*DisplayPtr*, *Procedure***)) ( );**
**Display ****DisplayPtr*;
int (**Procedure***)( );**

FORTRAN Syntax

integer*4 fxsetafterfunction
external fxsetafterfunction
integer*4 *DisplayPtr*
integer*4 *Procedure*
integer*4 *ReturnCode*
ReturnCode **= fxsetafterfunction(***DisplayPtr*, *Procedure***)**

Description

The **XSetAfterFunction** subroutine sets which subroutine is to be called following the execution of a subroutine that generates a protocol request. The specified procedure is called with a display pointer only.

Parameters

Displayptr	Specifies the connection to the X Server.
Procedure	Specifies the subroutine to be called.

Error Code

BadImplementation

Implementation Specifics

This Xlib subroutine is part of AIXwindows Run Time Environment in AIXwindows Environment/6000.

XSetArcMode Subroutine

Purpose

Sets the arc mode of a specified graphics context.

Libraries

Enhanced X-Windows Library (**libX11.a**)

FORTRAN 77 Library (**libXfx.a**)

C Syntax

XSetArcMode(*DisplayPtr*, *GraphicsContext*, *ArcMode*);
Display **DisplayPtr*;
GC *GraphicsContext*;
int *ArcMode*;

FORTRAN Syntax

external fxsetarcmode
integer*4 *DisplayPtr*
integer*4 *GraphicsContext*
integer*4 *ArcMode*
call fxsetarcmode(*DisplayPtr*, *GraphicsContext*, *ArcMode*)

Description

The **XSetArcMode** subroutine sets the arc mode in a specified graphics context. Arcs are filled with either chord or pie-slice patterns.

Parameters

DisplayPtr	Specifies the connection to the X Server.
GraphicsContext	Specifies the graphics context.
ArcMode	Specifies the arc mode as the value of either **ArcChord** or **ArcPieSlice**.

Error Codes

BadAlloc

BadGC

BadImplementation

BadValue

Implementation Specifics

This Xlib subroutine is part of AIXwindows Run Time Environment in AIXwindows Environment/6000.

Related Information

The **ChangeGC** protocol request.

XSetBackground Subroutine

Purpose

Sets the background for a specified graphics context.

Libraries

Enhanced X-Windows Library (**libX11.a**)

FORTRAN 77 Library (**libXfx.a**)

C Syntax

XSetBackground(*DisplayPtr*, *GraphicsContext*, *Background*);
Display **DisplayPtr*;
GC *GraphicsContext*;
unsigned long *Background*;

FORTRAN Syntax

external fxsetbackground
integer*4 *DisplayPtr*
integer*4 *GraphicsContext*
integer*4 *Background*
call fxsetbackground(*DisplayPtr*, *GraphicsContext*, *Background*)

Description

The **XSetBackground** subroutine sets the background color for the specified graphics context.

Parameters

DisplayPtr	Specifies the connection to the X Server.
GraphicsContext	Specifies the graphics context.
Background	Specifies the background color for a specified graphics context.

Error Codes

BadAlloc

BadGC

BadImplementation

Implementation Specifics

This Xlib subroutine is part of AIXwindows Run Time Environment in AIXwindows Environment/6000.

Related Information

The **ChangeGC** protocol request.

XSetClassHint Subroutine

Purpose

Sets the class of a window.

Libraries

Enhanced X-Windows Library (**libX11.a**)

FORTRAN 77 Library (**libXfx.a**)

C Syntax

XSetClassHint(*DisplayPtr*, *WindowID*, *ClassHints*);
Display **DisplayPtr*;
Window *WindowID*;
XClassHint **ClassHints*;

FORTRAN Syntax

external fxsetclasshint
integer*4 *DisplayPtr*
integer*4 *WindowID*, *ClassHints*
call fxsetclasshint(*DisplayPtr*, *WindowID*, *ClassHints*)

Description

The **XSetClassHint** subroutine sets the class hint for the specified window.

Parameters

DisplayPtr	Specifies the connection to the X Server.
WindowID	Specifies the window ID.
ClassHints	Specifies a pointer to a specified **XClassHint** data structure.

Error Codes

BadAlloc

BadImplementation

BadWindow

Implementation Specifics

This Xlib subroutine is part of AIXwindows Run Time Environment in AIXwindows Environment/6000.

XSetClipMask Subroutine

Purpose

Sets the clip mask of a specified graphics context to a specified pixmap.

Libraries

Enhanced X-Windows Library (**libX11.a**)

FORTRAN 77 Library (**libXfx.a**)

C Syntax

XSetClipMask(*DisplayPtr*, *GraphicsContext*, *PixmapID*);
Display **DisplayPtr*;
GC *GraphicsContext*;
Pixmap *PixmapID*;

FORTRAN Syntax

external fxsetclipmask
integer*4 *DisplayPtr*
integer*4 *GraphicsContext*
integer*4 *PixmapID*
call fxsetclipmask(*DisplayPtr*, *GraphicsContext*, *PixmapID*)

Description

The **XSetClipMask** subroutine sets the clip mask of a specified graphics context to a specified pixmap. If the *PixmapID* parameter is specified as a value of **None**, the pixels are always drawn regardless of the clip origin.

Parameters

DisplayPtr	Specifies the connection to the X Server.
GraphicsContext	Specifies the graphics context.
PixmapID	Specifies the ID of the pixmap or **None**.

Error Codes

BadAlloc

BadGC

BadImplementation

BadMatch

BadValue

Implementation Specifics

This Xlib subroutine is part of AIXwindows Run Time Environment in AIXwindows Environment/6000.

Related Information

The **ChangeGC** protocol request.

XSetClipOrigin Subroutine

Purpose

Sets the clip mask origin of a specified graphics context.

Libraries

Enhanced X-Windows Library (**libX11.a**)

FORTRAN 77 Library (**libXfx.a**)

C Syntax

XSetClipOrigin(*DisplayPtr*, *GraphicsContext*, *ClipXOrigin*, *ClipYOrigin)*;
Display **DisplayPtr*;
GC *GraphicsContext*;
int *ClipXOrigin*, *ClipYOrigin*;

FORTRAN Syntax

external fxsetcliporigin
integer*4 *DisplayPtr*
integer*4 *GraphicsContext*
integer*4 *ClipXOrigin*, *ClipYOrigin*
call fxsetcliporigin(*DisplayPtr*, *GraphicsContext*, *ClipXOrigin*, *ClipYOrigin*)

Description

The **XSetClipOrigin** subroutine sets the clip mask origin of a specified graphics context. The clip mask origin is relative to the origin of the destination drawable specified in the graphics protocol request.

Parameters

DisplayPtr	Specifies the connection to the X Server.
GraphicsContext	Specifies the graphics context.
ClipXOrigin	Specifies the x coordinate of the clip mask origin.
ClipYOrigin	Specifies the y coordinate of the clip mask origin.

Error Codes

BadAlloc

BadGC

BadImplementation

Implementation Specifics

This Xlib subroutine is part of AIXwindows Run Time Environment in AIXwindows Environment/6000.

Related Information

The **ChangeGC** protocol request.

XSetClipRectangles Subroutine

Purpose

Sets the clip mask of a specified graphics contexts to a specified list of rectangles.

Libraries

Enhanced X–Windows Library (**libX11.a**)

FORTRAN 77 Library (**libXfx.a**)

C Syntax

XSetClipRectangles(*DisplayPtr*, *GraphicsContext*,
ClipXOrigin, *ClipYOrigin*,
Rectangles, *Number*, *Ordering*);
Display **DisplayPtr*;
GC *GraphicsContext*;
int *ClipXOrigin*, *ClipOrigin*;
XRectangle *Rectangles*[];
int *Number*;
int *Ordering*;

FORTRAN Syntax

external fxsetcliprectangles
integer*4 *DisplayPtr*
integer*4 *GraphicsContext*
integer*4 *ClipXOrigin, ClipYOrigin*
integer*4 *Rectangles*
integer*4 *Number, Ordering*
call fxsetcliprectangles(*DisplayPtr, GraphicsContext, ClipXOrigin,*
ClipYOrigin, Rectangles, Number, Ordering)

Description

The **XSetClipRectangles** subroutine changes the *ClipMask* parameter in the specified graphics context to the specified list of rectangles.

The output is clipped to remain contained within the rectangles. The *ClipOrigin* parameter is interpreted relative to the origin of whatever destination drawable is specified in a graphics request. The rectangle coordinates are interpreted relative to the *ClipOrigin* parameter.

If the rectangles intersect, the graphics results are undefined. An empty array of rectangles disables output.

If known by the client, ordering relations on the rectangles can be specified with the *Ordering* parameter.

If incorrect ordering is specified, the X Server may generate a **BadMatch** error, but is not required to do so. On occasions when no error is generated the graphics results are undefined.

Parameters

DisplayPtr	Specifies the connection to the X Server.
GraphicsContext	Specifies the graphics context.

ClipXOrigin Specifies the X coordinate of the clip mask origin.

ClipYOrigin Specifies the Y coordinate of the clip mask origin.

Rectangles Specifies an array of rectangles that define the clip mask.

Number Specifies the number of rectangles.

Ordering Specifies the ordering of the rectangles. The *Ordering* parameter can be:

Unsorted Indicates that the rectangles are in arbitrary order.

YSorted Indicates that all rectangles are nondecreasing in their Y origin.

YXSorted Indicates that all rectangles are nondecreasing in their Y origin; and, rectangles with an equal Y origin are nondecreasing in their X origin.

YXBanded Indicates that all rectangles are nondecreasing in their Y origin; rectangles with an equal Y origin are nondecreasing in their X origin; and, for every possible Y scan line, all rectangles that include that scan line have an identical Y origin and Y extents.

Error Codes

BadAlloc

BadGC

BadImplementation

BadMatch

BadValue

Implementation Specifics

This Xlib subroutine is part of AIXwindows Run Time Environment in AIXwindows Environment/6000.

Related Information

The **SetClipRectangles** protocol request.

XSetCloseDownMode Subroutine

Purpose

Changes the close down mode of a client.

Libraries

Enhanced X-Windows Library (**libX11.a**)

FORTRAN 77 Library (**libXfx.a**)

C Syntax

XSetCloseDownMode(*DisplayPtr*, *CloseMode*);
Display **DisplayPtr*;
int *CloseMode*;

FORTRAN Syntax

external fxsetclosedownmode
integer*4 *DisplayPtr*
integer*4 *CloseMode*
call fxsetclosedownmode(*DisplayPtr*, *CloseMode*)

Description

The **XSetCloseDownMode** subroutine defines what happens to the client's resources at connection close.

The *CloseMode* parameter is set to the value of **DestroyAll** by default. It can be changed to the value of **RetainPermanent** or **RetainTemporary** by using the **XSetCloseDownMode** subroutine.

Parameters

DisplayPtr	Specifies the connection to the X Server.
CloseMode	Specifies the client close down mode as one of the following values:
	DestroyAll
	RetainPermanent
	RetainTemporary

Error Codes

BadImplementation

BadValue

Implementation Specifics

This Xlib subroutine is part of AIXwindows Run Time Environment in AIXwindows Environment/6000.

Related Information

The **SetCloseDownMode** protocol request.

XSetCommand Subroutine

Purpose

Sets the value of the command property.

Libraries

Enhanced X-Windows Library (**libX11.a**)

FORTRAN 77 Library (**libXfx.a**)

C Syntax

XSetCommand(*DisplayPtr*, *WindowID*,
argv, *argc*);
Display **DisplayPtr*;
Window *WindowID*;
char ***argv*;
int *argc*;

FORTRAN Syntax

external fxsetcommand
integer*4 *DisplayPtr*
integer*4 *WindowID*
integer*4 *argv*, *argc*
call fxsetcommand(*DisplayPtr*, *WindowID*, *argv*, *argc*)

Description

The **XSetCommand** subroutine sets the WM_COMMAND property.

Parameters

DisplayPtr	Specifies the connection to the X Server.
WindowID	Specifies the window ID.
argc	Specifies the number of parameters.
argv	Specifies the parameter list of an application. Typically, this parameter is the *argv* parameter array of your main program.

Error Codes

BadAlloc

BadImplementation

BadWindow

Implementation Specifics

This Xlib subroutine is part of AIXwindows Run Time Environment in AIXwindows Environment/6000.

Related Information

The **ChangeProperty** protocol request.

XSetDashes Subroutine

Purpose

Sets the dash offset and the dash list of dashed-line style for a specified graphics context.

Libraries

Enhanced X-Windows Library (**libX11.a**)

FORTRAN 77 Library (**libXfx.a**)

C Syntax

XSetDashes(*DisplayPtr*, *GraphicsContext*, *DashOffset*, *DashList*, *Number*);
Display **DisplayPtr*;
GC *GraphicsContext*;
int *DashOffset*;
char *DashList*[];
int *Number*;

FORTRAN Syntax

external fxsetdashes
integer*4 *DisplayPtr*
integer*4 *GraphicsContext*
integer*4 *DashOffset*, *DashList*, *Number*
call fxsetdashes(*DisplayPtr*, *GraphicsContext*, *DashOffset*, *DashList*, *Number*)

Description

The **XSetDashes** subroutine sets the *DashOffset* and *DashList* parameters for the dashed-line style in a specified graphics context. There must be at least one element specified for the *DashList* parameter. The initial and alternating elements of the *DashList* parameter are the even dashes, while the rest are the odd dashes. Specifying an odd-length list results in a list that concatenates with itself to produce an even-length list.

Each element of the *DashList* parameter specifies a dash length in pixels. All the elements must be nonzero.

The *DashOffset* parameter defines the phase of the pattern, specifying how many pixels into the dash list 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 value each time a cap-style is applied at a line end point.

The unit of measure for dashes is the same for 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. Otherwise, length should 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.

Parameters

DisplayPtr	Specifies the connection to the X Server.
GraphicsContext	Specifies the graphics context.
DashOffset	Specifies the phase of the pattern for the dashed-line style for the specified graphics context.
DashList	Specifies the dash list of the dashed-line style for the specified graphics context.
Number	Specifies the number of elements in the dash list.

Error Codes

BadAlloc

BadGC

BadImplementation

BadValue

Implementation Specifics

This Xlib subroutine is part of AIXwindows Run Time Environment in AIXwindows Environment/6000.

Related Information

The **SetDashes** protocol request.

XSetErrorHandler Subroutine

Purpose

Sets the error handler.

Libraries

Enhanced X-Windows Library (**libX11.a**)

FORTRAN 77 Library (**libXfx.a**)

C Syntax

XSetErrorHandler(*Handler***);**
int (**Handler***)(Display*, XErrorEvent*)**

FORTRAN Syntax

external fxseterrorhandler
integer*4 *Handler*
call fxseterrorhandler(*Handler***)**

Description

The **XSetErrorHandler** subroutine specifies the active error handler.

The **Xlib** library includes a default error handler to handle error events that are usually fatal, as well as one to handle error events from the X Server. The default error handlers print an explanatory message, and whichthen exit.

The **XSetErrorHandler** subroutine enables user-supplied routines for error handling. The error handler specification can be changed at any time.

The **XSetErrorHandler** subroutine calls the user-supplied error handler whenever an **XError** event is received. The error is handled as nonfatal. The error handler should not perform any direct or indirect operations on the display.

Parameter

Handler	Specifies the error handler. If **NULL**, the default error handler is invoked.

Error Code

BadImplementation

Implementation Specifics

This Xlib subroutine is part of AIXwindows Run Time Environment in AIXwindows Environment/6000.

Related Information

The **XErrorEvent** data structure

XSetFillRule Subroutine

Purpose

Sets the fill rule of a specified graphics context.

Libraries

Enhanced X-Windows Library (**libX11.a**)

FORTRAN 77 Library (**libXfx.a**)

C Syntax

XSetFillRule(*DisplayPtr*, *GraphicsContext*, *FillRule*);
Display **DisplayPtr*;
GC *GraphicsContext*;
int *FillRule*;

FORTRAN Syntax

external fxsetfillrule
integer*4 *DisplayPtr*
integer*4 *GraphicsContext*
integer*4 *FillRule*
call fxsetfillrule(*DisplayPtr*, *GraphicsContext*, *FillRule*)

Description

The **XSetFillRule** subroutine sets the fill rule in a specified graphics context.

Parameters

DisplayPtr	Specifies the connection to the X Server.
GraphicsContext	Specifies the graphics context.
FillRule	Specifies the fill rule as the value of **EvenOddRule** or the **WindingRule**.

Error Codes

BadAlloc

BadGC

BadImplementation

BadValue

Implementation Specifics

This Xlib subroutine is part of AIXwindows Run Time Environment in AIXwindows Environment/6000.

Related Information

The **ChangeGC** protocol request.

XSetFillStyle Subroutine

Purpose

Sets the fill style of a specified graphics context.

Libraries

Enhanced X-Windows Library (**libX11.a**)

FORTRAN 77 Library (**libXfx.a**)

C Syntax

XSetFillStyle(*DisplayPtr*, *GraphicsContext*, *FillStyle*);
Display **DisplayPtr*;
GC *GraphicsContext*;
int *FillStyle*;

FORTRAN Syntax

external fxsetfillstyle
integer*4 *DisplayPtr*
integer*4 *GraphicsContext*
integer*4 *FillStyle*
call fxsetfillstyle(*DisplayPtr*, *GraphicsContext*, *FillStyle*)

Description

The **XSetFillStyle** subroutine sets the fill style of a specified graphics context.

Parameters

DisplayPtr	Specifies the connection to the X Server.
GraphicsContext	Specifies the graphics context.
FillStyle	Specifies the fill style as one of the following values:

FillSolid **FillTiled**

FillStippled **FillOpaqueStippled**

Error Codes

BadAlloc

BadGC

BadImplementation

BadValue

Implementation Specifics

This Xlib subroutine is part of AIXwindows Run Time Environment in AIXwindows Environment/6000.

Related Information

The **ChangeGC** protocol request.

XSetFont Subroutine

Purpose

Sets the current font of a specified graphics context.

Libraries

Enhanced X-Windows Library (**libX11.a**)

FORTRAN 77 Library (**libXfx.a**)

C Syntax

XSetFont(*DisplayPtr*, *GraphicsContext*, *Font*);
Display **DisplayPtr*;
GC *GraphicsContext*;
Font *Font*;

FORTRAN Syntax

external fxsetfont
integer*4 *DisplayPtr*
integer*4 *GraphicsContext*
integer*4 *Font*
call fxsetfont(*DisplayPtr*, *GraphicsContext*, *Font*)

Description

The **XSetFont** subroutine sets the current font of a specified graphics context.

Parameters

DisplayPtr	Specifies the connection to the X Server.
GraphicsContext	Specifies the graphics context.
Font	Specifies the font ID.

Error Codes

BadAlloc

BadFont

BadGC

BadImplementation

Implementation Specifics

This Xlib subroutine is part of AIXwindows Run Time Environment in AIXwindows Environment/6000.

Related Information

The **ChangeGC** protocol request.

XSetFontPath Subroutine

Purpose

Sets the font search path.

Libraries

Enhanced X-Windows Library (**libX11.a**)

FORTRAN 77 Library (**libXfx.a**)

C Syntax

XSetFontPath(*DisplayPtr*, *Directories*, *NumberDirectories*);
Display ***DisplayPtr*;
char ****Directories*;
int *NumberDirectories*;

FORTRAN Syntax

external fxsetfontpath
integer*4 *DisplayPtr*
integer*4 *Directories*
integer*4 *NumberDirectories*
call fxsetfontpath(*DisplayPtr*, *Directories*, *NumberDirectories*)

Description

The **XSetFontPath** subroutine defines the directory search path for looking up fonts. There is one search path per X Server, not one per client. The contents of directory listings should not be used by client applications.

The X Server can usually cache font information internally, but flushes all cached information about fonts for which there are currently no resource IDs allocated.

An error from this request is system-specific.

Parameters

DisplayPtr	Specifies the connection to the X Server.
Directories	Specifies the directory search path. Setting the path to **NULL** restores the default path defined for the X Server.
NumberDirectories	Specifies the number of directories in the path.

Error Codes

BadImplementation

BadValue

Implementation Specifics

This Xlib subroutine is part of AIXwindows Run Time Environment in AIXwindows Environment/6000.

Related Information

The **XChar2b** data structure.

The **SetFontPath** protocol request.

XSetForeground Subroutine

Purpose

Sets the foreground color for a specified graphics context.

Libraries

Enhanced X-Windows Library (**libX11.a**)

FORTRAN 77 Library (**libXfx.a**)

C Syntax

XSetForeground(*DisplayPtr*, *GraphicsContext*, *Foreground*);
Display **DisplayPtr*;
GC *GraphicsContext*;
unsigned long *Foreground*;

FORTRAN Syntax

external fxsetforeground
integer*4 *DisplayPtr*
integer*4 *GraphicsContext*
integer*4 *Foreground*
call fxsetforeground(*DisplayPtr*, *GraphicsContext*, *Foreground*)

Description

The **XSetForeground** subroutine sets the foreground color for a specified graphics context.

Parameters

DisplayPtr	Specifies the connection to the X Server.
GraphicsContext	Specifies the graphics context.
Foreground	Specifies the foreground color for a specified graphics context.

Error Codes

BadAlloc

BadGC

BadImplementation

Implementation Specifics

This Xlib subroutine is part of AIXwindows Run Time Environment in AIXwindows Environment/6000.

Related Information

The **ChangeGC** protocol request.

XSetFunction Subroutine

Purpose

Sets the display function for a specified graphics context.

Libraries

Enhanced X-Windows Library (**libX11.a**)

FORTRAN 77 Library (**libXfx.a**)

C Syntax

XSetFunction(*DisplayPtr*, *GraphicsContext*, *Function*)
Display **DisplayPtr*;
GC *GraphicsContext*;
int *Function*;

FORTRAN Syntax

external fxsetfunction
integer*4 *DisplayPtr*
integer*4 *GraphicsContext*
integer*4 *Function*
call fxsetfunction(*DisplayPtr*, *GraphicsContext*, *Function*)

Description

The **XSetFunction** subroutine sets a specified display function in a specified graphics context.

Parameters

DisplayPtr	Specifies the connection to the X Server.
GraphicsContext	Specifies the graphics context.
Function	Specifies the display function to set for a specified graphics context.

Error Codes

BadAlloc

BadGC

BadImplementation

BadValue

Implementation Specifics

This Xlib subroutine is part of AIXwindows Run Time Environment in AIXwindows Environment/6000.

Related Information

The **ChangeGC** protocol request.

XSetGraphicsExposures Subroutine

Purpose

Sets the graphics-exposures flag of a specified graphics context.

Libraries

Enhanced X-Windows Library (**libX11.a**)

FORTRAN 77 Library (**libXfx.a**)

C Syntax

XSetGraphicsExposures(*DisplayPtr*, *GraphicsContext*, *GraphicsExposures*);
Display **DisplayPtr*;
GC *GraphicsContext*;
Boolean *GraphicsExposures*;

FORTRAN Syntax

external fxsetgraphicsexposures
integer*4 *DisplayPtr*
integer*4 *GraphicsContext*
integer*4 *GraphicsExposures*
call fxsetgraphicsexposures(*DisplayPtr*, *GraphicsContext*, *GraphicsExposures*)

Description

The **XSetGraphicsExposures** subroutine sets the graphics-exposures flag of a specified graphics context. It specifies if **GraphicsExpose** and **NoExpose** events are to be reported when using the **XCopyArea** and **XCopyPlane** subroutines with the specified graphics context.

Parameters

DisplayPtr	Specifies the connection to the X Server.
GraphicsContext	Specifies the graphics context.
GraphicsExposures	Specifies a Boolean value.

Return Values

True	Indicates that **GraphicsExpose** events are reported.
False	Indicates that **GraphicsExpose** events are not reported.

Error Codes

BadAlloc

BadGC

BadImplementation

BadValue

Implementation Specifics

This Xlib subroutine is part of AIXwindows Run Time Environment in AIXwindows Environment/6000.

Related Information

The **XCopyArea** subroutines, **XCopyPlane** subroutine.

The **ChangeGC** protocol request.

XSetIOErrorHandler Subroutine

Purpose

Handles fatal I/O errors.

Libraries

Enhanced X-Windows Library (**libX11.a**)

FORTRAN 77 Library (**libXfx.a**)

C Syntax

XSetIOErrorHandler (*Handler*)
int (**Handler*)(**Display ***);

FORTRAN Syntax

external fxsetioerrorhandler
integer*4 *Handler*
call fxsetioerrorhandler(*Handler*)

Description

The **XSetIOErrorHandler** subroutine sets the fatal I/O error handler. The **Xlib** library then calls the program-supplied error handler if a subroutine error occurs. This is handled by default as a fatal condition, and the subroutine should not return. If the I/O error handler does return, the client process exits.

Parameter

Handler	Specifies the program-supplied error handler.

Error Code

BadImplementation

Implementation Specifics

This Xlib subroutine is part of AIXwindows Run Time Environment in AIXwindows Environment/6000.

XSetIconName Subroutine

Purpose

Sets the name to be displayed in a specified window icon.

Library

Enhanced X-Windows Library (**libX11.a**)

FORTRAN 77 Library (**libXfx.a**)

C Syntax

XSetIconName(*DisplayPtr*, *WindowID*, *IconName*);
Display **DisplayPtr*;
Window *WindowID*;
char **IconName*;

FORTRAN Syntax

external fxseticonname
integer*4 *DisplayPtr*
integer*4 *WindowID*
character*256 *IconName*
call fxseticonname(*DisplayPtr*, *WindowID*, *IconName*)

Description

The **XSetIconName** subroutine sets the name to be displayed in a specified window icon. The name should be a null-terminated string.

Parameters

DisplayPtr	Specifies the connection to the X Server.
WindowID	Specifies the window ID.
IconName	Specifies the icon name.

Error Codes

BadAlloc

BadImplementation

BadWindow

Implementation Specifics

This Xlib subroutine is part of AIXwindows Run Time Environment in AIXwindows Environment/6000.

XSetIconSizes Subroutine

Purpose

Sets the icon size hints.

Libraries

Enhanced X-Windows Library (**libX11.a**)

FORTRAN 77 Library (**libXfx.a**)

C Syntax

XSetIconSizes(*DisplayPtr*, *WindowID*, *SizeList*, *Count*)
Display **DisplayPtr*;
Window *WindowID*;
XIconSize **SizeList*;
int *Count*;

FORTRAN Syntax

external fxseticonsizes
integer*4 *DisplayPtr*
integer*4 *WindowID*, *SizeList*, *Count*
call fxseticonsizes(*DisplayPtr*, *WindowID*, *SizeList*, *Count*)

Description

The **XSetIconSizes** subroutine, used only by window managers, sets supported icon sizes.

Parameters

DisplayPtr	Specifies the connection to the X Server.
WindowID	Specifies the window ID.
SizeList	Specifies a pointer to the size list.
Count	Specifies the number of items in the size list.

Error Codes

BadAlloc

BadImplementation

BadWindow

Implementation Specifics

This Xlib subroutine is part of AIXwindows Run Time Environment in AIXwindows Environment/6000.

Related Information

The **ChangeProperty** protocol request.

XSetInputFocus Subroutine

Purpose

Sets the input focus.

Libraries

Enhanced X-Windows Library (**libX11.a**)

FORTRAN 77 Library (**libXfx.a**)

C Syntax

XSetInputFocus(*DisplayPtr*, *Focus*,
RevertTo, *Timestamp*)
Display **DisplayPtr*;
Window *Focus*;
int *RevertTo*;
Time *Timestamp*;

FORTRAN Syntax

external fxsetinputfocus
integer*4 *DisplayPtr*
integer*4 *Focus*, *RevertTo*, *Timestamp*
call fxsetinputfocus(*DisplayPtr*, *Focus*, *RevertTo*, *Timestamp*)

Description

The **XSetInputFocus** subroutine changes the input focus and last-focus-change time. It has no effect if the specified time is earlier than the current last-focus-change time or later than the current X Server time. Otherwise, the last-focus-change time is set to the specified time and the **CurrentTime** value is replaced by the current X Server time. The **XSetInputFocus** subroutine generates the **FocusIn** and **FocusOut** events.

The specified focus window must be viewable at the time the **XSetInputFocus** subroutine is called. If, at some later time, the focus window is not viewable, the X Server evaluates the *RevertTo* parameter to determine the new focus window.

When the input focus reverts to a value of **RevertToPointerRoot** or **RevertToNone**, the X Server generates the **FocusIn** and **FocusOut** events, but the last-focus-change time is not affected.

Parameters

DisplayPtr	Specifies the connection to the X Server.	
Focus	Specifies the window ID for the input focus. The *Focus* parameter can specify one of the following values:	
	None	Indicates that all keyboard events are discarded until a new focus window is set. In this case, the *RevertTo* parameter is ignored.
	A window ID	Becomes the focus window for the keyboard. If a generated keyboard

event is usually reported to this window or to one of its inferiors, the event is reported normally. Otherwise, the event is reported relative to the focus window.

PointerRoot — Indicates that the focus window is the root window of whatever screen the pointer is on at each keyboard event. In this case, the *RevertTo* parameter is ignored.

RevertTo — Specifies an alternative for when the specified focus window is not viewable. The *RevertTo* parameter can specify one of the following values:

RevertToParent — The input focus reverts to the parent or to the closest viewable ancestor; the *RevertTo* parameter is reset to a **RevertToNone** value.

RevertToPointerRoot — The input focus reverts to a **PointerRoot** value.

RevertToNone — The input focus reverts to a value of **None**.

TimeStamp — Specifies the time as the **CurrentTime** value or as a time stamp expressed in milliseconds.

Error Codes

BadImplementation

BadMatch

BadValue

BadWindow

Implementation Specifics

This Xlib subroutine is part of AIXwindows Run Time Environment in AIXwindows Environment/6000.

Related Information

The **SetInputFocus** protocol request.

XSetLineAttributes Subroutine

Purpose

Sets the line-drawing components of a graphics context.

Libraries

Enhanced X-Windows Library (**libX11.a**)

FORTRAN 77 Library (**libXfx.a**)

C Syntax

XSetLineAttributes(*DisplayPtr*, *GraphicsContext*,
LineWidth, *LineStyle*,
CapStyle, *JoinStyle*)
Display **DisplayPtr*;
GC *GraphicsContext*;
unsigned int *LineWidth*;
int *LineStyle*;
int *CapStyle*;
int *JoinStyle*;

FORTRAN Syntax

external fxsetlineattributes
integer*4 *DisplayPtr*
integer*4 *GraphicsContext*
integer*4 *LineWidth*, *LineStyle*
integer*4 *CapStyle, JoinStyle*
call fxsetlineattributes(*DisplayPtr*, *GraphicsContext*, *LineWidth*,
LineStyle, *CapStyle*, *JoinStyle*)

Description

The **XSetLineAttributes** subroutine sets the line-drawing components in the specified graphics context.

Parameters

DisplayPtr	Specifies the connection to the X Server.
GraphicsContext	Specifies the graphics context.
LineWidth	Specifies the line width for the specified graphics context.
LineStyle	Specifies the line style as the following values:
	LineSolid
	LineOnOffDash
	LineDoubleDash.
CapStyle	Specifies the cap style as the following values:
	CapNotLast
	CapButt

	CapRound
	CapProjecting
JoinStyle	Specifies the line-join style as the following values:
	JoinMiter
	JoinRound
	JoinBevel

Error Codes

BadGC

BadImplementation

BadValue

Implementation Specifics

This Xlib subroutine is part of AIXwindows Run Time Environment in AIXwindows Environment/6000.

Related Information

The **ChangeGC** protocol request.

XSetModifierMapping Subroutine

Purpose

Sets the key codes to be used as modifiers.

Libraries

Enhanced X-Windows Library (**libX11.a**)

FORTRAN 77 Library (**libXfx.a**)

C Syntax

int XSetModifierMapping(*DisplayPtr*, *ModifierMap*);
Display **DisplayPtr*;
XModifierKeymap **ModifierMap*;

FORTRAN Syntax

integer*4 fxsetmodifiermapping
external fxsetmodifiermapping
integer*4 *DisplayPtr, ModifierMap*
integer*4 *ReturnCode*
ReturnCode = **fxsetmodifiermapping**(*DisplayPtr, ModiferMap*)

Description

The **XSetModifierMapping** subroutine specifies any keycodes to be used as modifiers. If it succeeds, the X Server generates a **MappingNotify** event, and the **XSetModifierMapping** subroutine returns a value of **MappingSuccess**. As many as eight modifier keys are permitted.

The *modifiermap* field of the **XModifierKeymap** data structure contains eight sets of *max_keypermod* key codes, one for each modifier in the **Shift**, **Lock**, **Control**, **Mod1**, **Mod2**, **Mod3**, **Mod4**, and **Mod5** order. Only nonzero key codes have meaning in each set; zero key codes are ignored. Also, nonzero keycodes must be in the range specified by the *min_keycode* and *max_keycode* fields of the **Display** data structure. No key code can be displayed twice.

An X Server can impose restrictions on how modifiers can be changed.

Parameters

DisplayPtr Specifies the connection to the X Server.

ModifierMap Specifies a pointer to the **XModifierKeymap** data structure.

Return Values

MappingBusy Indicates that the new keycodes specified for a modifier differ from those currently defined, and if any keys (current or new) for that modifier are in the logically down state, and none of the modifiers are changed.

MappingFailed Indicates that a restriction is disregarded, and none of the modifiers are changed.

MappingSuccess Indicates that the **XSetModifierMapping** subroutine succeeds.

Error Codes

BadAlloc

BadImplementation

BadValue

Implementation Specifics

This Xlib subroutine is part of AIXwindows Run Time Environment in AIXwindows Environment/6000.

Related Information

The **XModifierKeymap** data structure.

The **SetModifierMapping** protocol request.

XSetNormalHints Subroutine

Purpose

Sets size hints for a specified window in its normal state.

Libraries

Enhanced X-Windows Library (**libX11.a**)

FORTRAN 77 Library (**libXfx.a**)

C Syntax

void XSetNormalHints(*DisplayPtr*, *WindowID*, *Hints*)
Display **DisplayPtr*;
Window *WindowID*;
XSizeHints **Hints*;

FORTRAN Syntax

external fxsetnormalhints
integer*4 *DisplayPtr*
integer*4 *WindowID*
integer*4 *Hints*
call fxsetnormalhints(*DisplayPtr*, *WindowID*, *Hints*)

Description

The **XSetNormalHints** subroutine sets the size hints structure for a specified window. It allows an application to inform the window manager of the size or position desired for a window.

Also, an application can use **XSetNormalHints** subroutine along with direct calls to the **Xlib** library to resize or move its own window, since window managers may ignore redirected configure requests, but act on property changes.

To set size hints, an application must assign values to the appropriate members in the **XSizeHints** subroutine, and must set the *flags* field of the structure to indicate which information is present and where it came from.

Parameters

DisplayPtr	Specifies the connection to the X Server.
WindowID	Specifies the window ID.
Hints	Specifies a pointer to the sizing hints for the window in its normal state.

Error Codes

BadAlloc

BadImplementation

BadWindow

Implementation Specifics

This Xlib subroutine is part of AIXwindows Run Time Environment in AIXwindows Environment/6000.

Related Information

The **XSizeHints** data structure.

The **ChangeProperty** protocol request.

XSetPlaneMask Subroutine

Purpose

Sets the plane mask of a specified graphics context.

Libraries

Enhanced X-Windows Library (**libX11.a**)

FORTRAN 77 Library (**libXfx.a**)

C Syntax

XSetPlaneMask(*DisplayPtr*, *GraphicsContext*, *PlaneMask*);
Display **DisplayPtr*;
GC *GraphicsContext*;
unsigned long *PlaneMask*;

FORTRAN Syntax

external fxsetplanemask
integer*4 *DisplayPtr*
integer*4 *GraphicsContext*
integer*4 *PlaneMask*
call fxsetplanemask(*DisplayPtr*, *GraphicsContext*, *PlaneMask*)

Description

The **XSetPlaneMask** subroutine sets the plane mask of a specified graphics context.

Parameters

DisplayPtr	Specifies the connection to the X Server.
GraphicsContext	Specifies the graphics context.
PlaneMask	Specifies the plane mask.

Error Codes

BadAlloc

BadGC

BadImplementation

Implementation Specifics

This Xlib subroutine is part of AIXwindows Run Time Environment in AIXwindows Environment/6000.

Related Information

The **ChangeGC** protocol request.

XSetPointerMapping Subroutine

Purpose

Sets the mapping of the pointer buttons.

Libraries

Enhanced X-Windows Library (**libX11.a**)

FORTRAN 77 Library (**libXfx.a**)

C Syntax

int XSetPointerMapping(*DisplayPtr*, *Map*, *NumberMap*);
**Display ****DisplayPtr*;
unsigned char *Map*[];
int *NumberMap*;

FORTRAN Syntax

integer*4 fxsetpointermapping
external fxsetpointermapping
integer*4 *DisplayPtr*
integer*4 *Map*
integer*4 *NumberMap*
integer*4 *ReturnCode*
ReturnCode = **fxsetpointermapping(***DisplayPtr*, *Map*, *NumberMap*)

Description

The **XSetPointerMapping** subroutine sets the mapping of the pointer. If it succeeds, the X Server generates a **MappingNotify** event, and the **XSetPointerMapping** subroutine returns the value of **MappingSuccess**.

Elements of the mapping list are indexed starting from 1. The length of the list must be the same as would be returned by the **XGetPointerMapping** subroutine. The index is a core button number, and the element of the list defines the effective number. A 0 element disables a button. Elements are not restricted in value by the number of physical buttons. However, no two elements can have the same nonzero value. If any of the buttons to be altered are logically in the down state, the **XSetPointerMapping** subroutine returns the value of **MappingBusy**, and the mapping is not changed.

Parameters

DisplayPtr	Specifies the connection to the X Server.
Map	Specifies the mapping list.
NumberMap	Specifies the number of items in the mapping list.

Error Codes

BadImplementation

BadValue

Implementation Specifics

This Xlib subroutine is part of AIXwindows Run Time Environment in AIXwindows Environment/6000.

Related Information

Processing **MappingNotify** Events

The **SetPointerMapping** protocol request.

XSetRegion Subroutine

Purpose

Sets the clip-mask of a graphics context to a region.

Libraries

Enhanced X-Windows Library (**libX11.a**)

FORTRAN 77 Library (**libXfx.a**)

C Syntax

XSetRegion(*DisplayPtr*, *GraphicsContext*, *RegionPtr*)
Display **DisplayPtr*;
GC *GraphicsContext*;
Region *RegionPtr*;

FORTRAN Syntax

external fxsetregion
integer*4 *DisplayPtr*
integer*4 *GraphicsContext*, *RegionPtr*
call fxsetregion(*DisplayPtr*, *GraphicsContext*, *RegionPtr*)

Description

The **XSetRegion** subroutine sets the clip-mask in a graphics context to a specified region. Once the clip mask is set in the graphics context or once this subroutine completes and the region is no longer needed, the region can be deleted.

Parameters

DisplayPtr	Specifies the connection to the X Server.
GraphicsContext	Specifies the graphics context.
RegionPtr	Specifies the region to be used as the clip-mask.

Error Code

BadImplementation

Implementation Specifics

This Xlib subroutine is part of AIXwindows Run Time Environment in AIXwindows Environment/6000.

XSetScreenSaver Subroutine

Purpose

Sets the screen saver.

Libraries

Enhanced X-Windows Library (**libX11.a**)

FORTRAN 77 Library (**libXfx.a**)

C Syntax

XSetScreenSaver(*DisplayPtr*, *Timeout*, *Interval*,
PreferBlanking, *AllowExposures*)
Display **DisplayPtr*;
int *Timeout*, *Interval*;
int *PreferBlanking*;
int *AllowExposures*;

FORTRAN Syntax

external fxsetscreensaver
integer*4 *DisplayPtr*
integer*4 *Timeout*, *Interval*
integer*4 *PreferBlanking*, *AllowExposures*
call fxsetscreensaver(*DisplayPtr*, *Timeout*, *Interval*,
PreferBlanking, *AllowExposures*)

Description

The **XSetScreenSaver** subroutine sets the screen saver. The *Timeout* and *Interval* parameters are specified in seconds.

If the *Timeout* parameter value is nonzero, the screen saver is enabled. A value of 0 disables the screen saver, and a value of -1 restores the default. Other negative values generate an event error.

A value of 0 for the *Interval* parameter disables random pattern motion. Once the screen saver is enabled, if no device input is generated for the specified number of time out seconds, the screen saver is activated.

For each screen, if blanking is preferred and the hardware supports video blanking, the screen simply goes blank. Otherwise, if exposures are allowed or the screen can be regenerated without sending the **Expose** 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 by a subsequent device input or a call to the **XForceScreenSaver** subroutine set to the **ScreenSaverReset** mode.

Parameters

DisplayPtr	Specifies the connection to the X Server.
Timeout	Specifies the time out, in seconds, until the screen saver activates.
Interval	Specifies the interval between screen saver activities.

PreferBlanking	Specifies the enable screen blanking as the following values:
	DontPreferBlanking
	PreferBlanking
	DefaultBlanking
AllowExposures	Specifies the current screen saver control as the following values:
	DontAllowExposures
	AllowExposures
	DefaultExposures

Error Codes

BadImplementation

BadValue

Implementation Specifics

This Xlib subroutine is part of AIXwindows Run Time Environment in AIXwindows Environment/6000.

Related Information

The **SetScreenSaver** protocol request.

XSetSelectionOwner Subroutine

Purpose

Sets the selection owner.

Libraries

Enhanced X-Windows Library (**libX11.a**)

FORTRAN 77 Library (**libXfx.a**)

C Syntax

XSetSelectionOwner(*DisplayPtr*, *Selection*, *Owner*, *TimeStamp*)
Display **DisplayPtr*;
Atom *Selection*;
Window *Owner*;
Time *TimeStamp*;

FORTRAN Syntax

external fxsetselectionowner
integer*4 *DisplayPtr*
integer*4 *Selection*
integer*4 *Owner*
integer*4 *TimeStamp*
call fxsetselectionowner(*DisplayPtr*, *Selection*, *Owner*, *TimeStamp*)

Description

The **XSetSelectionOwner** subroutine sets the selection owner. It sets the last-change time to the value specified for the *TimeStamp* parameter; this value can be a time stamp expressed in milliseconds, or the value of **CurrentTime**, which is the current X Server time. However, it has no effect if the specified time is earlier than the last-change time of the specified selection or is later than the current X Server time.

If a value of **None** is specified for the *Owner* parameter, the selection will have no owner. Otherwise, the owner of the selection is 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 a value of **None**, the current owner is sent a **SelectionClear** event.

If the connection is subsequently closed for the owner of the specified window, the owner of the selection defaults to a value of **None**. The last-change time is not affected.

The X Server does not interpret the selection atom. The owner window is returned by the **XGetSelectionOwner** subroutine and is reported in the **SelectionRequest** and **SelectionClear** events. Selections are global to the X Server.

Parameters

DisplayPtr	Specifies the connection to the X Server.
Selection	Specifies the selection atom.
Owner	Specifies the owner of the specified selection atom as a window ID or as a value of **None**.

TimeStamp	Specifies the time as the value of **CurrentTime** or in a time stamp expressed in milliseconds.

Error Codes

BadAtom

BadImplementation

BadWindow

Implementation Specifics

This Xlib subroutine is part of AIXwindows Run Time Environment in AIXwindows Environment/6000.

Related Information

The **SetSelectionOwner** protocol request.

XSetSizeHints Subroutine

Purpose

Sets the values of a property of type WM_SIZE_HINTS.

Libraries

Enhanced X-Windows Library (**libX11.a**)

FORTRAN 77 Library (**libXfx.a**)

C Syntax

XSetSizeHints(*DisplayPtr*, *WindowID*, *Hints*,
Property);
Display **DisplayPtr*;
Window *WindowID*;
XSizeHints **Hints*;
Atom *Property*;

FORTRAN Syntax

external fxsetsizehints
integer*4 *DisplayPtr*
integer*4 *WindowID*, *Hints*, *Property*
call fxsetsizehints(*DisplayPtr*, *WindowID*, *Hints*, *Property*)

Description

The **XSetSizeHints** subroutine sets the **XSizeHints** data structure for a named property and specified window. These values are used by the **XSetNormalHints** and **XSetZoomHints** subroutines; they can be used to set the value of any property of type WM_SIZE_HINTS.

Parameters

DisplayPtr	Specifies the connection to the X Server.
WindowID	Specifies the window ID.
Hints	Specifies a pointer to the size hints.
Property	Specifies the property atom.

Error Codes

BadAlloc

BadAtom

BadImplementation

BadWindow

Implementation Specifics

This Xlib subroutine is part of AIXwindows Run Time Environment in AIXwindows Environment/6000.

Related Information

The **XSizeHints** data structure.

The **GetProperty** protocol request.

XSetStandardColormap Subroutine

Purpose

Sets a standard color map.

Libraries

Enhanced X-Windows Library (**libX11.a**)

FORTRAN 77 Library (**libXfx.a**)

C Syntax

void XSetStandardColormap(*DisplayPtr*, *WindowID*, *ColormapPtr*, *Property*)
Display **DisplayPtr*;
Window *WindowID*;
XStandardColormap **ColormapPtr*;
Atom *Property*;

FORTRAN Syntax

external fxsetstandardcolormap
integer*4 *DisplayPtr*
integer*4 *WindowID*, *ColormapPtr*, *Property*
call fxsetstandardcolormap(*DisplayPtr*, *WindowID*, *ColormapPtr*, *Property*)

Description

The **XSetStandardColormap** subroutine creates or changes a standard colormap.

The **XSetStandardColormap** subroutine is usually used only by window managers to create a standard colormap, according to the following procedure:

1. Opens a connection to the same server.
2. Grabs the server.
3. Sees if the *Property* parameter is on the property list of the root window for the screen.
4. If the desired property is not present:
   - Creates a colormap (not required for the RGB_DEFAULT_MAP property).
   - Determines the color capabilities of the display.
   - Calls either the **XAllocColorPlanes** or the **XAllocColorCells** subroutine to allocate cells in the colormap.
   - Calls the **XStoreColors** subroutine to store appropriate color values in the colormap.
   - Fills in the descriptive fields in the **XStandardColormap** data structure.
   - Attaches the property to the root window by calling the **XSetColormap** subroutine.
   - Calls the **XSetCloseDownMode** subroutine to make the resource permanent.
5. Ungrabs the server.

Parameters

Colormap	Specifies the colormap description.

DisplayPtr	Specifies the connection to the X Server.
Property	Specifies the property name to be set.
WindowID	Specifies the window ID.

Error Codes

BadAlloc

BadAtom

BadImplementation

BadWindow

Implementation Specifics

This Xlib subroutine is part of AIXwindows Run Time Environment in AIXwindows Environment/6000.

Related Information

The **XGrabServer** subroutine.

XSetStandardProperties Subroutine

Purpose

Specifies a minimum set of properties.

Libraries

Enhanced X-Windows Library (**libX11.a**)

FORTRAN 77 Library (**libXfx.a**)

C Syntax

XSetStandardProperties(*DisplayPtr*, *WindowID*,
WindowName,*IconName*,
IconPixmap, *argv*, *argc*, *Hints*)

Display **DisplayPtr*;
Window *WindowID*;
char **WindowName*;
char **IconName*;
Pixmap *IconPixmap*;
char ** *argv*;
int *argc*
XSizeHints **Hints*;

FORTRAN Syntax

external fxsetstandardproperties
integer*4 *DisplayPtr*
integer*4 *WindowID*
character*256 *WindowName*
character*256 *IconName*
integer*4 *IconPixmap*, *argv*, *argc*, *Hints*
call fxsetstandardproperties(*DisplayPtr*, *WindowID*,
WindowName, *IconName*,
IconPixmap, *argv*, *argc*, *Hints*)

Description

The **XSetStandardProperties** subroutine sets all or portions of the WM_NAME, WM_ICON, WM_ICON_NAME, WM_HINTS, WM_COMMAND, and WM_NORMAL_HINTS properties. It provides a way for simple applications to set essential properties with a single call. It should not be used by applications that need to communicate more information than is possible with this subroutine.

Parameters

argc	Specifies the number of arguments.
argv	Specifies the argument list of the application.
DisplayPtr	Specifies the connection to the X Server.
Hints	Specifies a pointer to the size hints for the window in its normal state.
IconName	Specifies the icon name .

IconPixmap	Specifies the icon as a pixmap or as **None**.
WindowID	Specifies the window ID.
WindowName	Specifies the window name as a null-terminated string.

Error Codes

BadAlloc

BadImplementation

BadWindow

Implementation Specifics

This Xlib subroutine is part of AIXwindows Run Time Environment in AIXwindows Environment/6000.

Related Information

The **ChangeProperty** protocol request.

XSetState Subroutine

Purpose

Sets the foreground, background, plane mask and function components in the graphics context.

Libraries

Enhanced X-Windows Library (**libX11.a**)

FORTRAN 77 Library (**libXfx.a**)

C Syntax

XSetState(*DisplayPtr*, *GraphicsContext*, *Foreground*,
Background, *Function*, *PlaneMask*)
Display **DisplayPtr*;
GC *GraphicsContext*;
unsigned long *Foreground*, *Background*;
int *Function*;
unsigned long *PlaneMask*;

FORTRAN Syntax

external fxsetstate
integer*4 *DisplayPtr*
integer*4 *GraphicsContext*
integer*4 *Foreground*, *Background*
integer*4 *Function*, *PlaneMask*
call fxsetstate(*DisplayPtr*, *GraphicsContext*, *Foreground*, *Background*,
Function, *PlaneMask*)

Description

The **XSetState** subroutine sets the foreground, background, plane mask, and function components in the specified graphics context.

Parameters

DisplayPtr	Specifies the connection to the X Server.
GraphicsContext	Specifies the graphics context.
Foreground	Specifies the foreground for the graphics context.
Background	Specifies the background.
Function	Specifies the function component.
PlaneMask	Specifies the plane mask.

Error Codes

BadGC

BadImplementation

BadValue

Implementation Specifics

This Xlib subroutine is part of AIXwindows Run Time Environment in AIXwindows Environment/6000.

Related Information

The **ChangeGC** protocol request.

XSetStipple Subroutine

Purpose

Sets the stipple of a specified graphics context.

Libraries

Enhanced X-Windows Library (**libX11.a**)

FORTRAN 77 Library (**libXfx.a**)

C Syntax

XSetStipple(*DisplayPtr*, *GraphicContext*, *Stipple***)**
Display **DisplayPtr*;
GC *GraphicsContext*;
Pixmap *Stipple*;

FORTRAN Syntax

external fxsetstipple
integer*4 *DisplayPtr*
integer*4 *GraphicsContext*
integer*4 *Stipple*
call fxsetstipple(*DisplayPtr*, *GraphicsContext*, *Stipple***)**

Description

The **XSetStipple** subroutine sets the stipple in a specified graphics context. The depth of the stipple must be 1.

Parameters

DisplayPtr	Specifies the connection to the X Server.
GraphicsContext	Specifies the graphics context.
Stipple	Specifies the stipple for the specified graphics context.

Error Codes

BadAlloc

BadGC

BadImplementation

BadMatch

BadPixmap

Implementation Specifics

This Xlib subroutine is part of AIXwindows Run Time Environment in AIXwindows Environment/6000.

Related Information

The **ChangeGC** protocol request.

XSetSubwindowMode Subroutine

Purpose

Sets the subwindow mode of a specified graphics context.

Libraries

Enhanced X-Windows Library (**libX11.a**)

FORTRAN 77 Library (**libXfx.a**)

C Syntax

XSetSubwindowMode(*DisplayPtr*, *GraphicsContext*, *SubwindowMode*)
Display **DisplayPtr*;
GC *GraphicsContext*;
int *SubwindowMode*;

FORTRAN Syntax

external fxsetsubwindowmode
integer*4 *DisplayPtr*
integer*4 *GraphicsContext*
integer*4 *SubwindowMode*
call fxsetsubwindowmode(*DisplayPtr*, *GraphicsContext*, *SubwindowMode*)

Description

The **XSetSubwindowMode** subroutine sets the subwindow mode of a specified graphics context

Parameters

DisplayPtr	Specifies the connection to the X Server.
GraphicsContext	Specifies the graphics context.
SubwindowMode	Specifies the subwindow mode as the value of **ClipByChildren** or **IncludeInferiors.**

Error Codes

BadAlloc

BadGC

BadImplementation

BadValue.

Implementation Specifics

This Xlib subroutine is part of AIXwindows Run Time Environment in AIXwindows Environment/6000.

Related Information

The **ChangeGC** protocol request.

XSetTSOrigin Subroutine

Purpose

Sets the tile or stipple origin of the specified graphics context.

Libraries

Enhanced X-Windows Library (**libX11.a**)

FORTRAN 77 Library (**libXfx.a**)

C Syntax

XSetTSOrigin(*DisplayPtr*, *GraphicsContext*, *TSXOrigin*, *TSYOrigin*)
Display **DisplayPtr*;
GC *GraphicsContext*;
int *TSXOrigin*, *TSYOrigin*;

FORTRAN Syntax

external fxsettsorigin
integer*4 *DisplayPtr*
integer*4 *GraphicsContext*
integer*4 *TSXOrigin*, *TSYOrigin*
call fxsettsorigin(*DisplayPtr*, *GraphicsContext*, *TSXOrigin*, *TSYOrigin*)

Description

The **XSetTSOrigin** subroutine sets the tile or stipple origin of the specified graphics context.

Parameters

DisplayPtr	Specifies the connection to the X Server.
GraphicsContext	Specifies the graphics context.
TSXOrigin	Specifies the x coordinate of the tile or stipple origin for the specified graphics context.
TSYOrigin	Specifies the y coordinate of the tile or stipple origin for the specified graphics context.

Error Codes

BadAlloc

BadGC

BadImplementation

Implementation Specifics

This Xlib subroutine is part of AIXwindows Run Time Environment in AIXwindows Environment/6000.

Related Information

The **ChangeGC** protocol request.

XSetTile Subroutine

Purpose

Sets the fill tile of a specified graphics context.

Libraries

Enhanced X-Windows Library (**libX11.a**)

FORTRAN 77 Library (**libXfx.a**)

C Syntax

XSetTile(*DisplayPtr*, *GraphicsContext*, *Tile*)
Display **DisplayPtr*;
GC *GraphicsContext*;
Pixmap *Tile*;

FORTRAN Syntax

external fxsettile
integer*4 *DisplayPtr*
integer*4 *GraphicsContext*
integer*4 *Tile*
call fxsettile(*DisplayPtr*, *GraphicsContext*, *Tile*)

Description

The **XSetTile** subroutine sets the fill tile of a specified graphics context. The depth of the tile must be the same as the depth of the screen.

Parameters

DisplayPtr	Specifies the connection to the X Server.
GraphicsContext	Specifies the graphics context.
Tile	Specifies the fill tile for the specified graphics context.

Error Codes

BadAlloc

BadGC

BadImplementation

BadMatch

BadPixmap

Implementation Specifics

This Xlib subroutine is part of AIXwindows Run Time Environment in AIXwindows Environment/6000.

Related Information

The **ChangeGC** protocol request.

XSetTransientForHint Subroutine

Purpose

Sets the WM_TRANSIENT_FOR property for a window.

Libraries

Enhanced X-Windows Library (**libX11.a**)

FORTRAN 77 Library (**libXfx.a**)

C Syntax

XSetTransientForHint(*DisplayPtr*, *WindowID*, *PropertyWindow***)**
Display **DisplayPtr*;
Window *WindowID*;
Window *PropertyWindow*;

FORTRAN Syntax

external fxsettransientforhint
integer*4 *DisplayPtr*
integer*4 *WindowID*, *PropertyWindow*
call fxsettransientforhint(*DisplayPtr*, *WindowID*, *PropertyWindow***)**

Description

The **XSetTransientForHint** subroutine sets the WM_TRANSIENT_FOR property of a specified window to a specified *PropertyWindow* parameter. It indicates to the window manager that a transient, top-level window is operating on behalf of another window, as when a dialog box is transient for the window of an application. Some window managers can use this information to unmap an application's dialog boxes; this may be desirable, for example, when the main application window gets iconified.

Parameters

DisplayPtr	Specifies the connection to the X Server.
WindowID	Specifies the window ID.
PropertyWindow	Specifies the window ID for which the WM_TRANSIENT_FOR property is to be set.

Error Codes

BadAlloc

BadImplementation

BadWindow

Implementation Specifics

This Xlib subroutine is part of AIXwindows Run Time Environment in AIXwindows Environment/6000.

XSetWMHints Subroutine

Purpose

Sets the window manager hints for a specified window.

Libraries

Enhanced X-Windows Library (**libX11.a**)

FORTRAN 77 Library (**libXfx.a**)

C Syntax

XSetWMHints(*DisplayPtr*, *WindowID*, *WMHintsPtr***)**
Display ******DisplayPtr*;
Window *WindowID*;
XWMHints ******WMHintsPtr*;

FORTRAN Syntax

external fxsetwmhints
integer*4 *DisplayPtr*
integer*4 *WindowID*
integer*4 *WmHintsPtr*
call fxsetwmhints(*DisplayPtr*, *WindowID*, *WmHintsPtr***)**

Description

The **XSetWMHints** subroutine sets the window manager hints, which includes icon information and location, the initial state of the window, and whether the application relies on the window manager to get keyboard input.

Parameters

DisplayPtr	Specifies the connection to the X Server.
WindowID	Specifies the window ID.
WMHintsPtr	Specifies a pointer to the window manager hints.

Error Codes

BadAlloc

BadImplementation

BadWindow

Implementation Specifics

This Xlib subroutine is part of AIXwindows Run Time Environment in AIXwindows Environment/6000.

Related Information

The **XWMHints** data structure.

The **XGetWMHints** subroutine.

The **ChangeProperty** protocol.

XSetWindowBackground Subroutine

Purpose

Sets the background of a window to a specified pixel.

Libraries

Enhanced X-Windows Library (**libX11.a**)

FORTRAN 77 Library (**libXfx.a**)

C Syntax

XSetWindowBackground(*DisplayPtr*, *WindowID*, *BackgroundPixel*)
Display **DisplayPtr*;
Window *WindowID*;
unsigned long *BackgroundPixel*;

FORTRAN Syntax

external fxsetwindowbackground
integer*4 *DisplayPtr*
integer*4 *WindowID*
integer*4 *BackgroundPixel*
call fxsetwindowbackground(*DisplayPtr*, *WindowID*, *BackgroundPixel*)

Description

The **XSetWindowBackground** subroutine sets the background pixel of a window to a specified pixel value. Changing the background does not cause the window contents to change. The **XSetWindowBackground** subroutine uses a pixmap of undefined size filled with the pixel value specified in the *BackgroundPixel* parameter.

The **XSetWindowBackground** subroutine cannot be used to change the background of an **InputOnly** window.

Parameters

DisplayPtr	Specifies the connection to the X Server.
WindowID	Specifies the window ID.
BackgroundPixel	Specifies the pixel to be used for the background.

Error Codes

BadImplementation

BadMatch

BadWindow

Implementation Specifics

This Xlib subroutine is part of AIXwindows Run Time Environment in AIXwindows Environment/6000.

Related Information

The **ChangeWindowAttributes** protocol request.

XSetWindowBackgroundPixmap Subroutine

Purpose

Sets the background of a window to a specified pixmap.

Libraries

Enhanced X-Windows Library (**libX11.a**)

FORTRAN 77 Library (**libXfx.a**)

C Syntax

XSetWindowBackgroundPixmap(*DisplayPtr*, *WindowID*, *BackgroundPixmap*);
Display **DisplayPtr*;
Window *WindowID*;
Pixmap *BackgroundPixmap*;

FORTRAN Syntax

external fxsetwindowbackgroundpixmap
integer*4 *DisplayPtr*
integer*4 *WindowID*
integer*4 *BackgroundPixmap*
call fxsetwindowbackgroundpixmap(*DisplayPtr*, *WindowID*, *BackgroundPixmap*)

Description

The **XSetWindowBackgroundPixmap** subroutine sets the background pixmap of a window to a specified pixmap. Changing the background does not cause the window contents to be changed. The background pixmap can be freed immediately if no further references to it will be made.

If the value of **ParentRelative** is specified for the *BackgroundPixmap* parameter, the background pixmap of the parent of the window is used; on the root window, the default background is restored. If a value of **None** is specified for the *BackgroundPixmap* parameter, the window has no defined background.

The **XSetWindowBackgroundPixmap** subroutine cannot be used on an **InputOnly** window.

Parameters

DisplayPtr	Specifies the connection to the X Server.
WindowID	Specifies the window ID.
BackgroundPixmap	Specifies the background pixmap as the value of **ParentRelative** or a value of **None**.

Error Codes

BadColor

BadImplementation

BadMatch

BadPixmap

BadWindow

Implementation Specifics

This Xlib subroutine is part of AIXwindows Run Time Environment in AIXwindows Environment/6000.

Related Information

The **ChangeWindowAttributes** protocol request.

XSetWindowBorder Subroutine

Purpose

Changes and repaints the border of a window to a specified pixel.

Libraries

Enhanced X-Windows Library (**libX11.a**)

FORTRAN 77 Library (**libXfx.a**)

C Syntax

XSetWindowBorder(*DisplayPtr*, *WindowID*, *BorderPixel*);
Display **DisplayPtr*;
Window *WindowID*;
unsigned long *BorderPixel*;

FORTRAN Syntax

external fxsetwindowborder
integer*4 *DisplayPtr*
integer*4 *WindowID*
integer*4 *BorderPixel*
call fxsetwindowborder(*DisplayPtr*, *WindowID*, *BorderPixel*)

Description

The **XSetWindowBorder** subroutine sets the border pixel of the window to the pixel value specified.

The **XSetWindowBorder** subroutine cannot be used on an **InputOnly** window.

Parameters

DisplayPtr	Specifies the connection to the X Server.
WindowID	Specifies the window ID.
BorderPixel	Specifies the entry in the colormap.

Error Codes

BadImplementation

BadMatch

BadWindow

Implementation Specifics

This Xlib subroutine is part of AIXwindows Run Time Environment in AIXwindows Environment/6000.

Related Information

The **XChangeWindowAttributes** subroutine, **XSetWindowBorderPixmap** subroutine

The **ChangeWindowAttributes** protocol request.

XSetWindowBorderPixmap Subroutine

Purpose

Changes and repaints the border tile of a specified window.

Libraries

Enhanced X-Windows Library (**libX11.a**)

FORTRAN 77 Library (**libXfx.a**)

C Syntax

XSetWindowBorderPixmap(*DisplayPtr*, *WindowID*, *BorderPixmap*)
Display **DisplayPtr*;
Window *WindowID*;
Pixmap *BorderPixmap*;

FORTRAN Syntax

external fxsetwindowborderpixmap
integer*4 *DisplayPtr*
integer*4 *WindowID*
integer*4 *BorderPixmap*
call fxsetwindowborderpixmap(*DisplayPtr*, *WindowID*, *BorderPixmap*)

Description

The **XSetWindowBorderPixmap** subroutine sets the border pixmap of the window to the pixmap specified. The border pixmap can be freed immediately if no further references to it will be made.

If the value of **CopyFromParent** is specified for the *BorderPixmap* parameter, a copy of the border pixmap of the parent window is used.

The **XSetWindowBorderPixmap** subroutine cannot be used on an **InputOnly** window.

Parameters

DisplayPtr	Specifies the connection to the X Server.
WindowID	Specifies the window ID.
BorderPixmap	Specifies the border pixmap.

Error Codes

BadImplementation

BadMatch

BadPixmap

BadWindow

Implementation Specifics

This Xlib subroutine is part of AIXwindows Run Time Environment in AIXwindows Environment/6000.

Related Information

The **ChangeWindowAttributes** protocol request.

The **XChangeWindowAttributes** subroutine, **XSetWindowBorder** subroutine, **XSetWindowBorderWidth** subroutine.

XSetWindowBorderWidth Subroutine

Purpose

Changes the border width of a specified window.

Libraries

Enhanced X-Windows Library (**libX11.a**)

FORTRAN 77 Library (**libXfx.a**)

C Syntax

XSetWindowBorderWidth(*DisplayPtr*, *WindowID*, *Width***)**
**Display ****DisplayPtr*;
Window *WindowID*;
unsigned int *Width*;

FORTRAN Syntax

external fxsetwindowborderwidth
integer*4 *DisplayPtr*
integer*4 *WindowID*
integer*4 *Width*
call fxsetwindowborderwidth(*DisplayPtr*, *WindowID*, *Width***)**

Description

The **XSetWindowBorderWidth** subroutine sets the width of a specified window border.

Parameters

DisplayPtr	Specifies the connection to the X Server.
WindowID	Specifies the window ID.
Width	Specifies the width for the window border.

Error Codes

BadImplementation

BadWindow

Implementation Specifics

This Xlib subroutine is part of AIXwindows Run Time Environment in AIXwindows Environment/6000.

Related Information

The **XChangeWindowAttributes** subroutine, **XSetWindowBorder** subroutine, **XSetWindowBorderPixmap** subroutine.

The **ConfigureWindow** protocol.

XSetWindowColormap Subroutine

Purpose

Sets the colormap of a specified window.

Libraries

Enhanced X-Windows Library (**libX11.a**)

FORTRAN 77 Library (**libXfx.a**)

C Syntax

XSetWindowColormap(*DisplayPtr*, *WindowID*, *ColormapID*)
Display **DisplayPtr*;
Window *WindowID*;
Colormap *ColormapID*;

FORTRAN Syntax

external fxsetwindowcolormap
integer*4 *DisplayPtr*
integer*4 *WindowID*, *ColormapID*
call fxsetwindowcolormap(*DisplayPtr*, *WindowID*, *ColormapID*)

Description

The **XSetWindowColormap** subroutine sets the colormap for a specified window. The colormap must have the same visual type as the window, or a **BadMatch** error results..

Parameters

DisplayPtr	Specifies the connection to the X Server.
WindowID	Specifies the window ID.
ColormapID	Specifies the colormap ID.

Error Codes

BadColor

BadImplementation

BadMatch

BadWindow

Implementation Specifics

This Xlib subroutine is part of AIXwindows Run Time Environment in AIXwindows Environment/6000.

Related Information

The **XChangeWindowAttributes** subroutine.

The **ChangeWindowAttributes** protocol request.

XSetZoomHints Subroutine

Purpose

Sets the value of the zoom hints for a window.

Libraries

Enhanced X-Windows Library (**libX11.a**)

FORTRAN 77 Library (**libXfx.a**)

C Syntax

XSetZoomHints(*DisplayPtr*, *WindowID*, *ZoomHints*)
Display **DisplayPtr*;
Window *WindowID*;
XSizeHints **ZoomHints*;

FORTRAN Syntax

external fxsetzoomhints
integer*4 *DisplayPtr*
integer*4 *WindowID*
integer*4 *ZoomHints*
call fxsetzoomhints(*DisplayPtr*, *WindowID*, *ZoomHints*)

Description

The **XSetZoomHints** subroutine provides the window manager with information for the window in the zoomed state by setting the WM_ZOOM_HINTS property.

Parameters

DisplayPtr	Specifies the connection to the X Server.
WindowID	Specifies the window ID.
ZoomHints	Specifies a pointer to the zoom hints.

Error Codes

BadAlloc

BadImplementation

BadWindow

Implementation Specifics

This Xlib subroutine is part of AIXwindows Run Time Environment in AIXwindows Environment/6000.

Related Information

The **XSizeHints** data structure.

The **XGetNormalHints** subroutine, **XGetSizeHints** subroutine, **XGetZoomHints** subroutine, **XSetNormalHints** subroutine, **XSetSizeHints** subroutine.

The **ChangeProperty** protocol.

XShrinkRegion Subroutine

Purpose

Reduces or enlarges a region by a specified amount.

Libraries

Enhanced X-Windows Library (**libX11.a**)

FORTRAN 77 Library (**libXfx.a**)

C Syntax

XShrinkRegion(*RegionPtr*, *DestinationX*, *DestinationY*)
Region *RegionPtr*;
int *DestinationX*, *DestinationY*;

FORTRAN Syntax

external fxshrinkregion
integer*4 *RegionPtr*, *DestinationX*, *DestinationY*
call fxshrinkregion(*RegionPtr*, *DestinationX*, *DestinationY*)

Description

The **XShrinkRegion** subroutine reduces or enlarges a region by a specified amount. Positive values reduce the size of the region; negative values increase the size of the region.

Parameters

RegionPtr	Specifies the region.
DestinationX	Specifies the x coordinate for the amount by which to reduce or enlarge the specified region.
DestinationY	Specifies the y coordinate for the amount by which to reduce or enlarge the specified region.

Error Code

BadImplementation

Implementation Specifics

This Xlib subroutine is part of AIXwindows Run Time Environment in AIXwindows Environment/6000.

Related Information

The **XCreateRegion** subroutine, **XDestroyRegion** subroutine, **XOffsetRegion** subroutine.

XStoreBuffer Subroutine

Purpose

Stores data in a specified cut buffer.

Libraries

Enhanced X-Windows Library (**libX11.a**)

FORTRAN 77 Library (**libXfx.a**)

C Syntax

XStoreBuffer(*DisplayPtr*, *Bytes*, *NumberBytes*, *Buffer*)
Display **DisplayPtr*;
char **Bytes*;
int *NumberBytes*;
int *Buffer*;

Fortran Syntax

external fxstorebuffer
integer*4 *DisplayPtr*
integer*4 *Bytes*
integer*4 *NumberBytes*, *Buffer*
call fxstorebuffer(*DisplayPtr*, *Bytes*, *NumberBytes*, *Buffer*)

Description

The **XStoreBuffer** subroutine stores data in a specified cut buffer. The data to be stored does not have to be null-terminated or an ASCII string.

Parameters

DisplayPtr	Specifies the connection to the X Server.
Bytes	Specifies the bytes to be stored.
NumberBytes	Specifies the number of bytes to be stored.
Buffer	Specifies the buffer in which to store the string.

Error Codes

BadAlloc
BadAtom
BadImplementation

Implementation Specifics

This Xlib subroutine is part of AIXwindows Run Time Environment in AIXwindows Environment/6000.

Related Information

The **XFetchBuffer** subroutine, **XFetchBytes** subroutine, **XStoreBytes** subroutine.

The **ChangeProperty** protocol.

XStoreBytes Subroutine

Purpose

Stores data in cut buffer zero.

Libraries

Enhanced X-Windows Library (**libX11.a**)

FORTRAN 77 Library (**libXfx.a**)

C Syntax

XStoreBytes(*DisplayPtr*, *Bytes*, *NumberBytes*)
Display **DisplayPtr*;
char **Bytes*;
int *NumberBytes*;

FORTRAN Syntax

external fxstorebytes
integer*4 *DisplayPtr*
integer*4 *Bytes*
integer*4 *NumberBytes*
call fxstorebytes(*DisplayPtr*, *Bytes*, *NumberBytes*)

Description

The **XStoreBytes** subroutine stores data in cut buffer 0. The data to be stored does not have to be null-terminated or an ASCII string.

The cut buffer contents can be retrieved later by any client with the **XFetchBytes** subroutine.

Parameters

DisplayPtr	Specifies the connection to the X Server.
Bytes	Specifies the bytes to be stored.
NumberBytes	Specifies the number of bytes to be stored.

Error Codes

BadAlloc

BadImplementation

Implementation Specifics

This Xlib subroutine is part of AIXwindows Run Time Environment in AIXwindows Environment/6000.

Related Information

The **XFetchBytes** subroutine.

The **ChangeProperty** protocol.

XStoreColor Subroutine

Purpose

Stores an RGB (red, green, and blue) value into a single colormap cell.

Libraries

Enhanced X-Windows Library (**libX11.a**)

FORTRAN 77 Library (**libXfx.a**)

C Syntax

XStoreColor(*DisplayPtr*, *ColormapID*, *Definitions*);
Display **DisplayPtr*;
Colormap *ColormapID*;
XColor **Definitions*;

FORTRAN Syntax

external fxstorecolor
integer*4 *DisplayPtr*
integer*4 *ColormapID*
integer*4 *Definitions*
call fxstorecolor(*DisplayPtr*, *ColormapID*, *Definitions*)

Description

The **XStoreColor** subroutine changes the colormap entry of the pixel value specified in the pixel field of the **XColor** data structure. This pixel value must be a read–write cell and a valid index into the colormap. If it is not a valid index into the colormap, a **BadValue** error is generated. The **XStoreColor** subroutine also changes the RGB color components. The components to be changed are specified by setting the **DoRed, DoGreen,** and/or **DoBlue** values in the *flags* field of the **XColor** data structure. If the colormap is an installed map for its screen, the changes are visible immediately.

Parameters

DisplayPtr	Specifies the connection to the X Server.
ColormapID	Specifies the colormap ID.
Definitions	Specifies the pointer to the color definitions structure. This contains the pixel and RGB values.

Error Codes

BadAccess

BadColor

BadImplementation

BadValue

Implementation Specifics

This Xlib subroutine is part of AIXwindows Run Time Environment in AIXwindows Environment/6000.

Related Information

The **XColors** data structure.

The **XStoreColors** subroutine.

The **StoreColors** protocol.

XStoreColors Subroutine

Purpose

Stores multiple RGB (red, green, and blue) values into colormap cells.

Libraries

Enhanced X-Windows Library (**libX11.a**)

FORTRAN 77 Library (**libXfx.a**)

C Syntax

XStoreColors(*DisplayPtr*, *ColormapID*, *Definitions*,
NumberColors);
Display **DisplayPtr*;
Colormap *ColormapID*;
XColor *Definitions*[];
int *NumberColors*;

FORTRAN Syntax

external fxstorecolors
integer*4 *DisplayPtr*
integer*4 *ColormapID*
integer*4 *Definitions, NumberColors*
call fxstorecolors(*DisplayPtr*, *ColormapID*, *Definitions*,
NumberColors)

Description

The **XStoreColors** subroutine changes the colormap entries of the pixel values specified in the *pixel field* of each of the members of the **XColor** data structure array.

The components to be changed are specified by setting the **DoRed**, **DoGreen**, and **DoBlue** values in the *flags* field in each of the **XColor** data structures. If the colormap is an installed map for its screen, the color changes are visible immediately.

The **XStoreColors** subroutine changes the specified pixels that are allocated writable in the *ColormapID* parameter by any client, even if one or more pixels is not a valid index into the colormap. If more than one pixel is not a valid index into the colormap, it is arbitrary which one will be reported.

Parameters

DisplayPtr	Specifies the connection to the X Server.
ColormapID	Specifies the colormap ID.
Definitions	Specifies an array of color definition structures.
NumberColors	Specifies the number of **XColor** data structures in the color definition array.

Error Codes

BadAccess

BadColor

BadImplementation

BadValue

Implementation Specifics

This Xlib subroutine is part of AIXwindows Run Time Environment in AIXwindows Environment/6000.

Related Information

The **XColors** data structure.

The **XStoreColor** subroutine.

The **StoreColors** protocol request.

XStoreName Subroutine

Purpose

Assigns a name to a window.

Libraries

Enhanced X-Windows Library (**libX11.a**)

FORTRAN 77 Library (**libXfx.a**)

C Syntax

XStoreName(*DisplayPtr*, *WindowID*, *WindowName*)
Display **DisplayPtr*;
Window *WindowID*;
char **WindowName*;

FORTRAN Syntax

external fxstorename
integer*4 *DisplayPtr*
integer*4 *WindowID*
character*256 *WindowName*
call fxstorename(*DisplayPtr*, *WindowID*, *WindowName*)

Description

The **XStoreName** subroutine assigns the name specified in the *WindowName* parameter to a specified window. This name is returned in subsequent calls to the **XFetchName** subroutine.

A window manager can display the window name in a prominent place, such as the title bar, so users can identify windows easily. The window name can also be displayed in the window icon, although using an existing window icon name is recommended.

Parameters

DisplayPtr	Specifies the connection to the X Server.
WindowID	Specifies the window ID.
WindowName	Specifies the window name as a null-terminated string.

Error Codes

BadAlloc

BadImplementation

BadWindow

Implementation Specifics

This Xlib subroutine is part of AIXwindows Run Time Environment in AIXwindows Environment/6000.

Related Information

The **XFetchName** subroutine, **XGetIconName** subroutine, **XSetIconName** subroutine.

The **ChangeProperty** protocol.

XStoreNamedColor Subroutine

Purpose

Sets the color of a pixel to a named color.

Libraries

Enhanced X-Windows Library (**libX11.a**)

FORTRAN 77 Library (**libXfx.a**)

C Syntax

XStoreNamedColor(*DisplayPtr*, *ColormapID*,
ColorName, *Pixel*, *Flags*)
Display **DisplayPtr*;
Colormap *ColormapID*;
char **ColorName*;
unsigned long *Pixel*;
int *Flags*;

FORTRAN Syntax

external fxstorenamedcolor
integer*4 *DisplayPtr*
integer*4 *ColormapID*
character*256 *ColorName*
integer*4 *Pixel*, *Flags*
call fxstorenamedcolor(*DisplayPtr*, *ColormapID*, *ColorName*,
Pixel, *Flags*)

Description

The **XStoreNamedColor** subroutine looks up the named color for the screen associated with the colormap and stores the result in the specified colormap.

The specified pixel must be a valid index into the colormap, and must be allocated. It must not, however, be allocated as read-only.

The *Flags* parameter specifies which RGB (red, green, and blue) components are set. This parameter can be set to the bitwise inclusive OR of the bits from the constant set to the values of **DoRed**, **DoGreen**, and **DoBlue**.

The **XStoreNamedColor** subroutine employs the ISO Latin-1 encoding; it is not case-sensitive.

Parameters

DisplayPtr	Specifies the connection to the X Server.
ColormapID	Specifies the colormap ID.
ColorName	Specifies the color name string.
Flags	Specifies which RGB components are set.
Pixel	Specifies the entry in the colormap.

Error Codes

BadAccess

BadColor

BadImplementation

BadName

BadValue

Implementation Specifics

This Xlib subroutine is part of AIXwindows Run Time Environment in AIXwindows Environment/6000.

Related Information

The **XFreeColors** subroutine, **XStoreColor** subroutine, **XStoreColors** subroutine.

The **StoreNamedColor** protocol request.

XStringToKeysym Subroutine

Purpose

Converts the name of a key symbol to the key symbol code.

Libraries

Enhanced X-Windows Library (**libX11.a**)

FORTRAN 77 Library (**libXfx.a**)

C Syntax

KeySym XStringToKeysym(*KeySymName*)
char **KeySymName*;

FORTRAN Syntax

integer*4 fxstringtokeysym
external fxstringtokeysym
character*256 *KeySymName*
integer*4 *Status*
Status = **fxstringtokeysym**(*KeySymName*)

Description

The **XStringToKeysym** subroutine converts the key symbol name to the key symbol code. Valid key symbol names are listed in the **<X11/keysymdef.h>** data file by removing the `XK_` prefix from each name. If the specified string does not match a valid key symbol, the **XStringToKeysym** subroutine returns the value of **NoSymbol**.

Parameter

KeySymName Specifies the name of the key symbol to be converted.

Return Values

NoSymbol The specified string does not match a valid key symbol.

Symbol The symbol that matches the specified string.

Error Code

BadImplementation

Implementation Specifics

This Xlib subroutine is part of AIXwindows Run Time Environment in AIXwindows Environment/6000.

Related Information

The **XKeycodeToKeysym** subroutine, **XKeysymToString** subroutine, **XKeysymToKeycode** subroutine, **XRebindKeysym** subroutine.

XSubImage Subroutine

Purpose

Creates a subimage.

Libraries

Enhanced X-Windows Library (**libX11.a**)

FORTRAN 77 Library (**libXfx.a**)

C Syntax

XImage *XSubImage(*XImagePtr, X, Y, SubImageWidth,* **SubImageHeight)**
XImage * *XImagePtr*;
int *X*;
int *Y*;
int *SubImageWidth*;
int *SubImageHeight*;

FORTRAN Syntax

integer*4 fxsubimage
external fxsubimage
integer*4 *XImagePtr, X, Y, SubImageWidth, SubImageHeight*
integer*4 *SubImage*
SubImage **= fxsubimage(***XImagePtr, X, Y, SubImageWidth, SubImageHeight***)**

Description

The **XSubImage** subroutine creates a new image that is a subsection of an existing image. It allocates the memory necessary for the new **XImage** data structure and returns a pointer to the new image. The data is copied from the source image. The source image must contain the rectangle defined by the values for the *X, Y, SubImageWidth*, and *SubImageHeight* parameters. The **XSubImage** subroutine uses repetitive calls to the **XGetPixel** and **XPutPixel** subroutines.

Parameters

XImagePtr	Specifies a pointer to the image.
X	Specifies the x coordinates.
Y	Specifies the y coordinates.
SubImageWidth	Specifies the width of the new subimage, in pixels.
SubImageHeight	Specifies the height of the new subimage, in pixels.

Error Code

BadImplementation

Implementation Specifics

This Xlib subroutine is part of AIXwindows Run Time Environment in AIXwindows Environment/6000.

Related Information

The **XImage** data structure.

The **XAddPixel** subroutine, **XCreateImage** subroutine, **XGetPixel** subroutine, **XPutPixel** subroutine.

XSubtractRegion Subroutine

Purpose

Subtracts two regions.

Libraries

Enhanced X-Windows Library (**libX11.a**)

FORTRAN 77 Library (**libXfx.a**)

C Syntax

XSubtractRegion(*SourceA*, *SourceB*, *DestinationRegion*)
Region *SourceA*, *SourceB*, *DestinationRegion*;

FORTRAN Syntax

external fxsubtractregion
integer*4 *SourceA*, *SourceB*, *DestinationRegion*
call fxsubtractregion(*SourceA*, *SourceB*, *DestinationRegion*)

Description

The **XSubtractRegion** subroutine subtracts the region in the *SourceB* parameter from the region in the *SourceA* parameter, and then stores the result in the *DestinationRegion* parameter.

Parameters

DestinationRegion	Stores the result of the computation.
SourceA	Specifies one of the two regions for the computation.
SourceB	Specifies one of the two regions for the computation.

Error Code

BadImplementation

Implementation Specifics

This Xlib subroutine is part of AIXwindows Run Time Environment in AIXwindows Environment/6000.

Related Information

The **XIntersectRegion** subroutine, **XUnionRegion** subroutine, **XUnionRectWithRegion** subroutine, **XXorRegion** subroutine.

XSync Subroutine

Purpose

Flushes the output buffer and waits until all requests are completed.

Library

Enhanced X-Windows Library (**libX11.a**)

FORTRAN 77 Library (**libXfx.a**)

C Syntax

XSync(*DisplayPtr*, *Discard*)
Display **DisplayPtr*;
int *Discard*;

FORTRAN Syntax

external fxsync
integer*4 *DisplayPtr*, *Discard*
call fxsync(*DisplayPtr*, *Discard*)

Description

The subroutine **XSync** flushes the output buffer. Then, it waits until all requests 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, the **XSync** subroutine calls the **XError** subroutine. Any events generated by the server are enqueued into the library's event queue.

Client applications seldom need to call the **XSync** subroutine.

Parameters

DisplayPtr Specifies the connection to the X Server.

Discard Specifies whether to discard all events on the event queue. The *Discard* parameter can be one of the following values:

False Indicates that the **XSync** subroutine does not discard the events on the queue.

True Indicates that the **XSync** subroutine discards all events on the queue, including those events that were on the queue before it was called.

Error Code

BadImplementation

Implementation Specifics

This Xlib subroutine is part of AIXwindows Run Time Environment in AIXwindows Environment/6000.

Related Information

The **XFlush** subroutine.

The **GetInputFocus** protocol request.

XSynchronize Subroutine

Purpose

Enables or disables synchronization.

Libraries

Enhanced X-Windows Library (**libX11.a**)

FORTRAN 77 Library (**libXfx.a**)

C Syntax

int (*XSynchronize(*DisplayPtr*, *OnOff***))()**
**Display ****DisplayPtr*;
int *OnOff*;

FORTRAN Syntax

integer*4 fxsynchronize
external fxsynchronize
integer*4 *DisplayPtr*
integer*4 *OnOff*
integer*4 *ReturnCode*
ReturnCode = **fxsynchronize(***DisplayPtr*, *OnOff***)**

Description

The **XSynchronize** subroutine returns the previous after function. The **XSynchronize** subroutine enables or disables sychronization.

Parameters

DisplayPtr Specifies the connection to the X Server.

OnOff Specifies whether to enable or disable synchronization. The *OnOff* parameter can be either of the following values:

False If the **XSynchronize** subroutine disables synchronization or turns synchronization to off.

True If the **XSynchronize** subroutine sets synchronization to on.

Error Code

BadImplementation

Implementation Specifics

This Xlib subroutine is part of AIXwindows Run Time Environment in AIXwindows Environment/6000.

XTextExtents Subroutine

Purpose

Gets the bounding box of 1-byte character string.

Libraries

Enhanced X-Windows Library (**libX11.a**)

FORTRAN 77 Library (**libXfx.a**)

C Syntax

XTextExtents(*FontStructure*, *String*, *NumberCharacters*, *DirectionReturn*,
FontAscentReturn, *FontDescentReturn*, *OverallReturn*)
XFontStruct **FontStructure*;
char **String*;
int *NumberCharacters*;
int **DirectionReturn*;
int **FontAscentReturn*, **FontDescentReturn*;
XCharStruct **OverallReturn*;

FORTRAN Syntax

external fxtextextents
integer*4 *FontStructure*
character*256 *String*
integer*4 *NumberCharacters*, *DirectionReturn*, *FontAscentReturn*
integer*4 *FontDescentReturn*, *OverallReturn*
call fxtextextents(*FontStructure*, *String*, *NumberCharacters*, *DirectionReturn*,
FontAscentReturn, *FontDescentReturn*, *OverallReturn*)

Description

The **XTextExtents** subroutine 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. The **XTextExtents** subroutine performs the size computation locally.

The **XTextExtents** subroutine returns an **XCharStruct** structure with the *width* field set to the sum of the character-width metrics of all characters in the string. For each character in the string following should occur:

- Let *W* be the sum of the character-width metrics of all characters preceding it in the string.
- Let *R* be the right-side-bearing metric of the character plus *W*.
- The *lbearing* member is set to the the minimum *L* value of all characters in the string.
- The *rbearing* member is set to the maximum *R* value of all characters in the string.

Use the **XQueryTextExtents** subroutine to query the server for the sizes of an 8-bit character string.

Parameters

FontStructure	Specifies a pointer to the **XFontStruct** structure.
String	Specifies the character string.
NumberCharacters	Specifies the number of characters in the character string.
DirectionReturn	Returns the value of the direction (the value of **FontLeftToRight** or **FontRightToLeft**) hint member.
FontAscentReturn	Returns the font ascent member, which is the maximum of the ascent metrics of all characters in the string.
FontDescentReturn	Returns the font descent member, which is the maximum of the descent metrics.
OverallReturn	Returns the overall size in the specified **XCharStruct** structure.

Error Code

BadImplementation

Implementation Specifics

This Xlib subroutine is part of AIXwindows Run Time Environment in AIXwindows Environment/6000.

Related Information

The **XChar2b** data structure, **XCharStruct** data structure, **XFontStruct** data structure.

The **XQueryTextExtents** subroutine, **XQueryTextExtents16** subroutine, **XTextExtents16** subroutine.

XTextExtents16 Subroutine

Purpose

Gets the bounding box of 2-byte character string.

Libraries

Enhanced X-Windows Library (**libX11.a**)

FORTRAN 77 Library (**libXfx.a**)

C Syntax

XTextExtents16(*FontStructure*, *String*, *NumberCharacters*, *DirectionReturn*,
FontAscentReturn, *FontDescentReturn*, *OverallReturn*)
XFontStruct **FontStructure*;
XChar2b **String*;
int *NumberCharacters*;
int **DirectionReturn*;
int **FontAscentReturn*, **FontDescentReturn*;
XCharStruct **OverallReturn*;

FORTRAN Syntax

external fxtextextents16
integer*4 *FontStructure*
integer*4 *String*
integer*4 *NumberCharacters*, *DirectionReturn*, *FontAscentReturn*
integer*4 *FontDescentReturn*, *OverallReturn*
call fxtextextents16(*FontStructure*, *String*, *NumberCharacters*, *DirectionReturn*,
FontAscentReturn, *FontDescentReturn*, *OverallReturn*)

Description

The **XTextExtents16** subroutine returns the logical extents of the specified 2-byte character string. It performs the size computation locally.

The **XTextExtents16** subroutine returns an **XCharStruct** structure with the *width* field set to the sum of the character-width metrics of all characters in the string. For each character in the string the following should occur:

- Let *W* be the sum of the character-width metrics of all characters preceding it in the string.
- Let *R* be the right-side-bearing metric of the character plus the *W* variable value.
- The *lbearing* member is set to the minimum *L* value of all characters in the string.
- The *rbearing* member is set to the maximum *R* value of all characters in the string.

If the font has no defined default character, undefined characters in the string are a vlaue of **0**.

Use the **XQueryTextExtents16** subroutine to query the server for the sizes of a 16-bit character string.

Parameters

FontStructure	Specifies a pointer to the **XFontStruct** structure.
String	Specifies the character string.
NumberCharacters	Specifies the number of characters in the character string.
DirectionReturn	Returns the value of the direction (the value of **FontLeftToRight** or **FontRightToLeft**) hint member.
FontAscentReturn	Returns the font ascent member, which is the maximum of the ascent metrics of all characters in the string.
FontDescentReturn	Returns the font descent member, which is the maximum of the descent metrics.
OverallReturn	Returns the overall size in the specified **XCharStruct** structure.

Error Code

BadImplementation

Implementation Specifics

This Xlib subroutine is part of AIXwindows Run Time Environment in AIXwindows Environment/6000.

Related Information

The **XChar2b** data structure, **XCharStruct** data structure, **XFontStruct** data structure.

The **XQueryTextExtents** subroutine, **XQueryTextExtents16** subroutine, **XTextExtents** subroutine.

XTextWidth Subroutine

Purpose

Gets the width of an 8-bit character string.

Libraries

Enhanced X-Windows Library (**libX11.a**)

FORTRAN 77 Library (**libXfx.a**)

C Syntax

int XTextWidth(*FontStructure*, *String*, *Count***)**
XFontStruct **FontStructure*;
char **String*;
int *Count*;

FORTRAN Syntax

integer*4 fxtextwidth
external fxtextwidth
integer*4 *FontStructure*
character*256 *String*
integer*4 *Count*
integer*4 *Width8*
Width8 = **fxtextwidth(***FontStructure*, *String*, *Count***)**

Description

The **XTextWidth** subroutine determines the width of an 8-bit character string. The width is computed by adding the character widths of all of the characters. The **XTextWidth** subroutine returns the sum of the character metrics in pixels.

Parameters

FontSructure	Specifies the font used for the width computation.
String	Specifies the character string.
Count	Specifies the character count in the named string.

Error Code

BadImplementation

Implementation Specifics

This Xlib subroutine is part of AIXwindows Run Time Environment in AIXwindows Environment/6000.

Related Information

The **XChar2b** data structure, **XFontStruct** data structure.

The **XTextWidth16** subroutine.

XTextWidth16 Subroutine

Purpose

Gets the width of a 2-byte character string.

Libraries

Enhanced X-Windows Library (**libX11.a**)

FORTRAN 77 Library (**libXfx.a**)

C Syntax

int XTextWidth16(*FontStructure*, *String*, *Count***)**
XFontStruct **FontStructure*;
XChar2b **String*;
int *Count*;

FORTRAN Syntax

integer*4 fxtextwidth16
external fxtextwidth16
integer*4 *FontStructure*
integer*4 *String*
integer*4 *Count*
integer*4 *Width16*
Width16 = **fxtextwidth16(***FontStructure*, *String*, *Count***)**

Description

The **XTextWidth16** subroutine determines the width of a 2-byte character string. Width is computed by adding the character widths of all of the characters. The **XTextWidth16** subroutine returns the sum of the character metrics in pixels.

Parameters

FontStructure	Specifies the font used for the width computation.
String	Specifies the character string.
Count	Specifies the character count in the string.

Error Code

BadImplementation

Implementation Specifics

This Xlib subroutine is part of AIXwindows Run Time Environment in AIXwindows Environment/6000.

Related Information

The **XFontStruct** data structure, **XChar2b** data structure.

The **XTextWidth** subroutine.

XTranslateCoordinates Subroutine

Purpose

Transforms coordinates between windows.

Libraries

Enhanced X-Windows Library (**libX11.a**)

FORTRAN 77 Library (**libXfx.a**)

C Syntax

int XTranslateCoordinates(*DisplayPtr, SourceWindow, DestinationWindow, SourceX, SourceY, DestinationXReturn, DestinationYReturn, ChildReturn***)**
Display ****DisplayPtr*;**
Window *SourceWindow, DestinationWindow***;**
int *SourceX, SourceY***;**
int ****DestinationXReturn*, ****DestinationYReturn***;**
Window ****ChildReturn*;**

FORTRAN Syntax

integer*4 fxtranslatecoordinates
external fxtranslatecoordinates
integer*4 *DisplayPtr*
integer*4 *SourceWindow*
integer*4 *DestinationWindow*
integer*4 *SourceX, SourceY*
integer*4 *DestinationXReturn, DestinationYReturn*
integer*4 *ChildReturn*
integer*4 *ReturnCode*
ReturnCode = **fxtranslatecoordinates(***DisplayPtr, SourceWindow, DestinationWindow, SourceX, SourceY, DestinationXReturn, DestinationYReturn, ChildReturn***)**

Description

The **XTranslateCoordinates** subroutine performs a coordinate transformation from the coordinate space of one window to another window, or it determines which subwindow contains a coordinate.

The **XTranslateCoordinates** subroutine takes the *SourceX* and *SourceY* parameter 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 the *DestinationXReturn* and *DestinationYReturn* parameters.

If the **XTranslateCoordinates** subroutine returns a value of 0, it indicates that the *SourceWindow* and *DestinationWindow* parameters are on different screens and that the *DestinationXReturn* and *DestinationYReturn* parameters have a value of 0.

If the coordinates are contained in a mapped child window of the *DestinationWindow* parameter, that child window is returned to the *ChildReturn* parameter.

Parameters

ChildReturn	Returns the child window if the coordinates are contained in a mapped child of the destination window.
DestinationWindow	Specifies the window ID of the destination window.
DestinationXReturn	Returns the x coordinate within the destination window.
DestinationYReturn	Returns the y coordinate within the destination window.
DisplayPtr	Specifies the connection to the X Server.
SourceWindow	Specifies the window ID of the source window.
SourceX	Specifies the x coordinate within the source window.
SourceY	Specifies the y coordinate within the source window.

Return Values

False	The windows are not located on the same screen. The *DestinationXReturn* and the *DestinationYReturn* parameters have a value of 0.
True	The windows are located on the same screen.

Error Codes

BadImplementation

BadWindow

Implementation Specifics

This Xlib subroutine is part of AIXwindows Run Time Environment in AIXwindows Environment/6000.

Related Information

The **TranslateCoordinates** Protocol Request.

XUndefineCursor Subroutine

Purpose

Defines a cursor for a window.

Libraries

Enhanced X-Windows Library (**libX11.a**)

FORTRAN 77 Library (**libXfx.a**)

C Syntax

XUndefineCursor(*DisplayPtr, WindowID***);**
Display ****DisplayPtr*;**
Window *WindowID*;

FORTRAN Syntax

external fxundefinecursor
integer*4 *DisplayPtr*
integer*4 *WindowID*
call fxundefinecursor(*DisplayPtr, WindowID***)**

Description

The **XUndefineCursor** subroutine undefines the cursor in the window. When the mouse is in the window, the cursor of the parent window is used. When the root window cursor is undefined, the default cursor is restored.

Parameters

DisplayPtr	Specifies the connection to the X Server.
WindowID	Specifies the window ID.

Error Codes

BadImplementation

BadWindow

Implementation Specifics

This Xlib subroutine is part of AIXwindows Run Time Environment in AIXwindows Environment/6000.

Related Information

The **XDefineCursor** subroutine.

The **ChangeWindowAttributes** protocol.

XUngrabButton Subroutine

Purpose

Ungrabs a mouse button.

Libraries

Enhanced X-Windows Library (**libX11.a**)

FORTRAN 77 Library (**libXfx.a**)

C Syntax

XUngrabButton(*DisplayPtr*, *ButtonUngrab*, *Modifiers*, *UngrabWindow*);
Display **DisplayPtr*;
unsigned int *ButtonUngrab*;
unsigned int *Modifiers*;
Window *UngrabWindow*;

FORTRAN Syntax

external fxungrabbutton
integer*4 *DisplayPtr*
integer*4 *ButtonUngrab*
integer*4 *Modifiers*, *UngrabWindow*
call fxungrabbutton(*DisplayPtr*, *ButtonUngrab*, *Modifiers*, *UngrabWindow*)

Description

The **XUngrabButton** subroutine 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 an **XGrabButton** subroutine with the same button key combination on the same window.

The *ButtonUngrab* parameter can be set to the **AnyButton** value, which is equivalent to issuing the ungrab request for all possible buttons. This request has no effect on an active grab.

Parameters

DisplayPtr Specifies the connection to the X Server.

ButtonUngrab 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. The *Modifiers* parameter can be one of the following valid keymask bits:

ControlMask	**Mod2Mask**
LockMask	**Mod3Mask**
ShiftMask	**Mod4Mask**
Mod1Mask	**Mod5Mask**

AnyModifier	This 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.

UngrabWindow Specifies the window ID of the window to be ungrabbed.

Error Codes

BadImplementation

BadWindow

Implementation Specifics

This Xlib subroutine is part of AIXwindows Run Time Environment in AIXwindows Environment/6000.

Related Information

The **XChangeActivePointerGrab** subroutine, **XGrabButton** subroutine.

The **UngrabButton** Protocol Request.

XUngrabKey Subroutine

Purpose

Ungrabs a key.

Libraries

Enhanced X-Windows Library (**libX11.a**)

FORTRAN 77 Library (**libXfx.a**)

C Syntax

XUngrabKey(*DisplayPtr*, *Keycode*, *Modifiers*,
UngrabWindow);
Display **DisplayPtr*;
int *Keycode*;
unsigned int *Modifiers*;
Window *UngrabWindow*;

FORTRAN Syntax

external fxungrabkey
integer*4 *DisplayPtr*
integer*4 *Keycode*, *Modifiers*, *UngrabWindow*
call fxungrabkey(*DisplayPtr*, *Keycode*, *Modifiers*, *UngrabWindow*)

Description

The **XUngrabKey** subroutine 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.

Parameters

DisplayPtr Specifies the connection to the X Server.

Keycode Specifies the keycode or the value of **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. The *Modifiers* parameter can be set to the following valid keymask bits:

ShiftMask	**Mod2Mask**
LockMask	**Mod3Mask**
ControlMask	**Mod4Mask**
Mod1Mask	**Mod5Mask**

Or, it can be set to the value of **AnyModifier**, which is equivalent to issuing the ungrab key request for all possible modifier combinations.

UngrabWindow Specifies the window ID of the window associated with the keys to be ungrabbed.

Error Codes

BadImplementation

BadWindow

Implementation Specifics

This Xlib subroutine is part of AIXwindows Run Time Environment in AIXwindows Environment/6000.

Related Information

The **XGrabKey** subroutine, **XGrabKeyboard** subroutine, **XUngrabKeyboard** subroutine.

The **UngrabKey** Protocol Request.

XUngrabKeyboard Subroutine

Purpose

Ungrabs the keyboard.

Libraries

Enhanced X-Windows Library (**libX11.a**)

FORTRAN 77 Library (**libXfx.a**)

C Syntax

XUngrabKeyboard(*DisplayPtr*, *TimeStamp*);
Display **DisplayPtr*;
Time *TimeStamp*;

FORTRAN Syntax

external fxungrabkeyboard
integer*4 *DisplayPtr*
integer*4 *TimeStamp*
call fxungrabkeyboard(*DisplayPtr*, *TimeStamp*)

Description

The **XUngrabKeyboard** subroutine releases the keyboard and any queued events if the client has actively grabbed it with the **XGrabKeyboard** subroutine or the **XGrabKey** subroutine. If the specified time is earlier than the last-keyboard-grab time or is later than the current X Server time, the **XUngrabKeyboard** subroutine does not release the keyboard and any queued events.

The **XUngrabKeyboard** subroutine generates **FocusIn** and **FocusOut** events. The X Server automatically performs an **XUngrabKeyboard** subroutine if the event window for an active keyboard grab becomes unviewable.

Parameters

DisplayPtr Specifies the connection to the X Server.

TimeStamp Specifies the time in a time stamp, which is expressed in milliseconds, or the value of **CurrentTime**.

Error Code

BadImplementation

Implementation Specifics

This Xlib subroutine is part of AIXwindows Run Time Environment in AIXwindows Environment/6000.

Related Information

The **XGrabKey** subroutine, **XGrabKeyboard** subroutine, **XUngrabKey** subroutine.

The **UnGrabKeyboard** Protocol Request.

XUngrabPointer Subroutine

Purpose

Ungrabs the pointer.

Libraries

Enhanced X-Windows Library (**libX11.a**)

FORTRAN 77 Library (**libXfx.a**)

C Syntax

XUngrabPointer(*DisplayPtr, TimeStamp***)**
Display ****DisplayPtr*;**
Time *TimeStamp*;

FORTRAN Syntax

external fxungrabpointer
integer*4 *DisplayPtr*
integer*4 *TimeStamp*
call fxungrabpointer(*DisplayPtr, TimeStamp***)**

Description

The **XUngrabPointer** subroutine releases the pointer and any queued events, if this client has actively grabbed the pointer with the **XGrabPointer** or **XGrabButton** subroutines 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.

The **XUngrabPointer** subroutine 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** subroutine automatically.

Parameters

DisplayPtr	Specifies the connection to the X Server.
TimeStamp	Specifies the time in a time stamp, which is expressed in milliseconds, or the value of **CurrentTime**.

Error Codes

BadImplementation

BadWindow

Implementation Specifics

This Xlib subroutine is part of AIXwindows Run Time Environment in AIXwindows Environment/6000.

Related Information

The **XChangeActivePointerGrab** subroutine, **XGrabPointer** subroutine.

The **UngrabPointer** Protocol Request.

XUngrabServer Subroutine

Purpose

Ungrabs the server.

Libraries

Enhanced X-Windows Library (**libX11.a**)

FORTRAN 77 Library (**libXfx.a**)

C Syntax

XUngrabServer(*DisplayPtr***)**
**Display ****DisplayPtr*;

FORTRAN Syntax

external fxungrabserver
integer*4 *DisplayPtr*
call fxungrabserver(*DisplayPtr***)**

Description

The **XUngrabServer** subroutine restarts processing of requests and closedowns 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.

Parameter

DisplayPtr Specifies the connection to the X Server.

Error Code

BadImplementation

Implementation Specifics

This Xlib subroutine is part of AIXwindows Run Time Environment in AIXwindows Environment/6000.

Related Information

The **XGrabServer** subroutine.

The **UngrabServer** Protocol Request.

XUninstallColormap Subroutine

Purpose

Uninstalls a colormap.

Libraries

Enhanced X-Windows Library (**libX11.a**)

FORTRAN 77 Library (**libXfx.a**)

C Syntax

XUninstallColormap(*DisplayPtr*, *ColorMapID*);
Display **DisplayPtr*;
Colormap *ColorMapID*;

FORTRAN Syntax

external fxuninstallcolormap
integer*4 *DisplayPtr*
integer*4 *ColorMapID*
call fxuninstallcolormap(*DisplayPtr*, *ColoMapID*)

Description

The **XUninstallColormap** subroutine removes the specified colormap from the required list for its screen. As a result, the specified colormap 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 same *ColorMapID* resource ID. In addition, for every other colormap that is either installed or uninstalled as a result of a call to the **XUninstallColormap** subroutine, the X Server generates a **ColormapNotify** event on each window that has the same *ColorMapID* resource ID.

Parameters

DisplayPtr Specifies the connection to the X Server.

ColorMapID Specifies the colormap ID.

Error Codes

BadColor

BadImplementation

Implementation Specifics

This Xlib subroutine is part of AIXwindows Run Time Environment in AIXwindows Environment/6000.

Related Information

The **XInstallColormap** subroutine.

The **ColormapNotify** event subroutine.

The **UninstallColormap** Protocol Request.

XUnionRectWithRegion Subroutine

Purpose

Creates a union of a source region and a rectangle.

Libraries

Enhanced X-Windows Library (**libX11.a**)

FORTRAN 77 Library (**libXfx.a**)

C Syntax

XUnionRectWithRegion(*RectanglePtr*, *SourceRegion*,
DestinationRegionReturn);
Rectangle **RectanglePtr*;
Region *SourceRegion*;
Region *DestinationRegionReturn*;

FORTRAN Syntax

external fxunionrectwithregion
integer*4 *RectanglePtr*
integer*4 *SourceRegion*
integer*4 *DestinationRegionReturn*
call fxunionrectwithregion(*RectanglePtr*, *SourceRegion*, *DestinationRegionReturn*)

Description

The **XUnionRectWithRegion** subroutine updates the destination region from a union of the specified rectangle and the specified source region.

Parameters

RectanglePtr	Specifies the rectangle.
SourceRegion	Specifies the source region.
DestinationRegionReturn	Returns the destination region.

Error Code

BadImplementation

Implementation Specifics

This Xlib subroutine is part of AIXwindows Run Time Environment in AIXwindows Environment/6000.

Related Information

The **XIntersectRegion** subroutine, **XSubtractRegion** subroutine, **XUnionRegion** subroutine, **XXorRegion** subroutine.

XUnionRegion Subroutine

Purpose

Computes union of two regions.

Library

Enhanced X-Windows Library (**libX11.a**)

FORTRAN 77 Library (**libXfx.a**)

C Syntax

XUnionRegion(*SourceA, SourceB, DestinationRegion***)**
Region *SourceA, SourceB, DestinationRegion*;

FORTRAN Syntax

external fxunionregion
integer*4 *SourceA, SourceB, DestinationRegion*
call fxunionregion(*SourceA, SourceB, DestinationRegion***)**

Description

The **XUnionRegion** subroutine computes the union of two regions.

Parameters

SourceA, SourceB	Specifies the two regions for the computation.
DestinationRegion	Stores the result of the computation.

Error Code

BadImplementation

Implementation Specifics

This Xlib subroutine is part of AIXwindows Run Time Environment in AIXwindows Environment/6000.

XUniqueContext Subroutine

Purpose

Creates a new context.

Libraries

Enhanced X-Windows Library (**libX11.a**)

C Syntax

XContext XUniqueContext()

Description

The **XUniqueContext** subroutine creates a unique context type that can be used in subsequent calls to the **XSaveContext** subroutine and the **XFindContext** subroutine.

Error Code

BadImplementation

Implementation Specifics

This Xlib subroutine is part of AIXwindows Run Time Environment in AIXwindows Environment/6000.

XUnloadFont Subroutine

Purpose

Unloads the specified font.

Libraries

Enhanced X-Windows Library (**libX11.a**)

FORTRAN 77 Library (**libXfx.a**)

C Syntax

XUnloadFont(*DisplayPtr*, *FontID*)
Display **DisplayPtr*;
Font *FontID*;

FORTRAN Syntax

external fxunloadfont
integer*4 *DisplayPtr*
integer*4 *FontID*
call fxunloadfont(*DisplayPtr*, *FontID*)

Description

The **XUnloadFont** subroutine unloads the specified font loaded by the **XLoadFont** subroutine. 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 should not be referenced again.

Parameters

DisplayPtr	Specifies the connection to the X Server.
FontID	Specifies the font ID.

Error Codes

BadFont

BadImplementation

Implementation Specifics

This Xlib subroutine is part of AIXwindows Run Time Environment in AIXwindows Environment/6000.

Related Information

The **Char2b** data structure.

The **CloseFont** Protocol Request.

XUnmapSubwindows Subroutine

Purpose

Unmaps all subwindows for a specified window.

Libraries

Enhanced X-Windows Library (**libX11.a**)

FORTRAN 77 Library (**libXfx.a**)

C Syntax

XUnmapSubwindows(*DisplayPtr, WindowID***)**
Display ****DisplayPtr*;**
Window *WindowID*;

FORTRAN Syntax

external fxunmapsubwindows
integer*4 *DisplayPtr*
integer*4 *WindowID*
call fxunmapsubwindows(*DisplayPtr, WindowID***)**

Description

The **XUnmapSubwindows** subroutine unmaps all subwindows at one time for a specified window. Subwindows are unmapped in bottom-to-top stacking order.

The X Server generates an **UnmapNotify** event on each subwindow and an **Expose** event on formerly obscured windows.

Parameters

DisplayPtr	Specifies the connection to the X Server.
WindowID	Specifies the window ID.

Error Codes

BadImplementation

BadWindow

Implementation Specifics

This Xlib subroutine is part of AIXwindows Run Time Environment in AIXwindows Environment/6000.

Related Information

The **UnmapSubwindows** Protocol Request.

XUnmapWindow Subroutine

Purpose

Unmaps a specified window.

Libraries

Enhanced X-Windows Library (**libX11.a**)

FORTRAN 77 Library (**libXfx.a**)

C Syntax

XUnmapWindow(*DisplayPtr*, *WindowID*)
Display **DisplayPtr*;
Window *WindowID*;

FORTRAN Syntax

external fxunmapwindow
integer*4 *DisplayPtr*
integer*4 *WindowID*
call fxunmapwindow(*DisplayPtr*, *WindowID*)

Description

The **XUnmapWindow** subroutine unmaps a specified window. The X Server generates an **UnmapNotify** event. Unmapping a window generates the **Expose** events on formerly obscured windows.

If the specified window is already unmapped, the **XUnmapWindow** subroutine has no effect. Normal exposure processing on formerly obscured windows is performed. Any child window will no longer be visible until another map call is made on the parent. That is, the subwindows are still mapped but are not visible until the parent window is mapped.

Parameters

DisplayPtr	Specifies the connection to the X Server.
WindowID	Specifies the window ID.

Error Codes

BadImplementation

BadWindow

Implementation Specifics

This Xlib subroutine is part of AIXwindows Run Time Environment in AIXwindows Environment/6000.

Related Information

The **UnmapWindow** Protocol Request.

XUseKeymap Subroutine

Purpose

Changes keymap files.

Libraries

Enhanced X-Windows Library (**liboldX.a**)

FORTRAN 77 Library (**libXfx.a**)

C Syntax

Status XUseKeymap(*KeymapFile***)**
char ****KeymapFile*;**

FORTRAN Syntax

external fxusekeymap
integer*4 fxusekeymap
integer*4 *Status*
integer*4 *KeymapFile*
status = fxusekeymap(*KeymapFile***)**

Description

The **XUseKeymap** subroutine provides an alternate keymap file for the **XLookupMapping** subroutine. It changes the keymap file. This change only affects the keymap within the current process. If the **XUseKeymap** subroutine is unsuccessful, the existing keymap is untouched.

Parameter

KeymapFile	Specifies the name of the keymap file to use with the current process.

Return Values

One	The **XUseKeymap** subroutine succeeds.
Zero	The **XUseKeymap** subroutine cannot find the keymap file named **keymapfile,** or if the file contains a bad magic number.

Error Code

BadImplementation

Implementation Specifics

This Xlib subroutine is part of AIXwindows Run Time Environment in AIXwindows Environment/6000.

XVisualIDFromVisual Subroutine

Purpose

Gets the visual ID for a specified visual type.

Libraries

Enhanced X-Windows Library (**libX11.a**)

FORTRAN 77 Library (**libXfx.a**)

C Syntax

VisualID XVisualIDFromVisual(*VisualPtr***)**
Visual ****VisualPtr*;**

FORTRAN Syntax

external fxvisualidfromvisual
integer*4 *VisualPtr*
integer*4 *DisplayPtr*
VisualPtr = **fxvisualidfromvisual(***DisplayPtr***)**

Description

The **XVisualIDFromVisual** subroutine returns the visual ID for the specified visual type.

Parameter

VisualPtr	Specifies the visual type.

Error Code

BadImplementation

Implementation Specifics

This Xlib subroutine is part of AIXwindows Run Time Environment in AIXwindows Environment/6000.

XWarpPointer Subroutine

Purpose

Moves the pointer to arbitrary point on the screen.

Libraries

Enhanced X-Windows Library (**libX11.a**)

FORTRAN 77 Library (**libXfx.a**)

C Syntax

XWarpPointer(*DisplayPtr, SourceWindow, DestinationWindow,*
SourceX, SourceY, SourceWidth, SourceHeight,
DestinationX, DestinationY)
Display **DisplayPtr*;
Window *SourceW, DestinationW*;
int *SourceX, SourceY*;
unsigned int *SourceWidth, SourceHeight*;
int *DestinationX, DestinationY*;

FORTRAN Syntax

external fxwarppointer
integer*4 *DisplayPtr*
integer*4 *SourceWindow, DestinationWindow*
integer*4 *SourceX, SourceY*
integer*4 *SourceWidth, SourceHeight*
integer*4 *DestinationX, DestinationY*
call fxwarppointer(*DisplayPtr, SourceWindow, DestinationWindow, SourceX,*
SourceY, SourceWidth, SourceHeight,
DestinationX, DestinationY)

Description

The **XWarpPointer** subroutine moves the pointer to an arbitrary point on the screen.

- If the *DestinationWindow* parameter has the value of **None**, the **XWarpPointer** subroutine moves the pointer by the *DestinationX* and *DestinationY* parameter offsets relative to the current position of the pointer. If the *DestinationWindow* parameter is a window, the **XWarpPointer** subroutine moves the pointer to the *DestinationX* and *DestinationY* parameter offsets relative to the origin of the *DestinationWindow* parameter.
- If the *SourceWindow* parameter has the value of **None**, the move is independent of the current position. If the *SourceWindow* parameter is a window ID, the move only takes place if the pointer is currently contained in a visible porion of the specified rectangle of the *SourceWindow* parameter.
- The *SourceX* and *SourceY* parameter coordinates are relative to the origin of the *SourceWindow* parameter. If the *SourceHeight* parameter is the value of **0**, it is replaced with the current height of the *SourceWindow* parameter minus the *SourceY* parameter. If the *SourceWidth* parameter is a value of **0**, it is replaced with the current width of the *SourceWindow* parameter minus the *SourceX* parameter.

Normally, pointer movement control is left to the user. The **XWarpPointer** subroutine generates events just as if the user had instantaneously moved the pointer from one position to another.

Note: Do not use the **XWarpPointer** subroutine to move the pointer outside the *confine_to* window of an active pointer grab. An attempt to do so will only move the pointer as far as the closest edge of the *confine_to* window.

Parameters

DisplayPtr	Specifies the connection to the X Server.
SourceWindow	Specifies the window ID of the source window, or the value of **None**.
DestinationWindow	Specifies the window ID of the destination window, or the value of **None**.
SouceX	Specifies the x coordinate, which is relative to the origin of the source window.
SourceY	Specifies the y coordinate, which is relative to the origin of the source window.
SourceWidth	Specifies a rectangle in the source window.
SourceHeight	Specifies a rectangle in the source window.
DestinationX	Specifies the x coordinate of the destination window.
DestinationY	Specifies the y coordinate of the destination window.

Error Codes

BadImplementation

BadWindow

Implementation Specifics

This Xlib subroutine is part of AIXwindows Run Time Environment in AIXwindows Environment/6000.

Related Information

The **WarpPointer** Protocol Request.

XWindowEvent Subroutine

Purpose

Removes the next event that matches both a window and an event mask.

Libraries

Enhanced X-Windows Library (**libX11.a**)

FORTRAN 77 Library (**libXfx.a**)

C Syntax

XWindowEvent(*DisplayPtr*, *WindowID*, *EventMask*,
EventReturn)
Display **DisplayPtr*;
Window *WindowID*;
long *EventMask*;
XEvent **EventReturn*;

FORTRAN Syntax

external fxwindowevent
integer*4 *DisplayPtr*
integer*4 *WindowID*, *EventMask*, *EventReturn*
call fxwindowevent(*DisplayPtr*, *WindowID*, *EventMask*, *EventReturn*)

Description

The **XWindowEvent** subroutine searches the event queue for an event that matches both the specified window and event mask. When it finds a match, the **XWindowEvent** subroutine removes that event from the queue and copies it into the specified **XEvent** data structure.

The other events stored in the queue are not discarded. If a matching event is not in the queue, the **XWindowEvent** subroutine flushes the output buffer and blocks until one is received.

Parameters

DisplayPtr	Specifies the connection to the X Server.
WindowID	Specifies the Window ID.
EventMask	Specifies the event mask.
EventReturn	Returns the associated structure of the matched event.

Error Code

BadImplementation

Implementation Specifics

This Xlib subroutine is part of AIXwindows Run Time Environment in AIXwindows Environment/6000.

XWriteBitmapFile Subroutine

Purpose

Writes out a bitmap to a file.

Libraries

Enhanced X-Windows Library (**libX11.a**)

FORTRAN 77 Library (**libXfx.a**)

C Syntax

int XWriteBitmapFile(*DisplayPtr, FileName, Bitmap, Width, Height, XHot, YHot***)**
Display ****DisplayPtr*;**
char ****FileName*;**
Pixmap *Bitmap*;
int *Width, Height*;
int *XHot, YHot*;

FORTRAN Syntax

integer*4 fxwritebitmapfile
external fxwritebitmapfile
integer*4 *DisplayPtr*
character*256 *FileName*
integer*4 *Bitmap, Width, Height, XHot, YHot*
integer*4 *ReturnCode*
ReturnCode = **fxwritebitmapfile(***DisplayPtr, FileName, Bitmap, Width, Height, XHot, YHot***)**

Description

The **XWriteBitmapFile** subroutine writes a bitmap to a file.

The default hot spot coordinates for the bitmap, if not specified in the *XHot* and *YHot* parameters, are –1, –1. Otherwise the **XWriteBitmapFile** subroutine writes out the specified values for the *XHot* and *YHot* parameters to the file as the the hotspot coordinates.

Note: The **XWriteBitmapFile** subroutine writes out X version 11 format only.

Parameters

DisplayPtr	Specifies the connection to the X Server.
FileName	Specifies the file name to use. The format for this file name is operating system dependent.
Bitmap	Specifies the bitmap to be written.
Width	Specifies the width of the bitmap.
Height	Specifies the height of the bitmap.
XHot	Specifies the hotspot coordinates; or, –1, –1, if not specified.
YHot	Specifies the hotspot coordinates; or, –1, –1, if not specified.

Return Values

BitmapOpenFailed	Indicates that the file cannot be opened for writing.
BitmapNoMemory	IIndicates that insufficient memory is allocated.
BitmapSuccess	IIndicates that no error occurs.

Error Codes

BadDrawable

BadImplementation

BadMatch

Implementation Specifics

This Xlib subroutine is part of AIXwindows Run Time Environment in AIXwindows Environment/6000.

Related Information

The **XReadBitmapFile** subroutine.

XXorRegion Subroutine

Purpose

Gets the difference between the union and the intersection of two regions.

Library

Enhanced X-Windows Library (**libX11.a**)

FORTRAN 77 Library (**libXfx.a**)

C Syntax

XXorRegion(*SourceA, SourceB, DestinationRegion*)
Region *SourceA*, *SourceB*, *DestinationRegion*;

FORTRAN Syntax

external fxxorregion
integer*4 *SourceA*, *SourceB*, *DestinationRegion*
call fxxorregion(*SourceA*, *SourceB*, *DestinationRegion*)

Description

The **XXorRegion** subroutine calculates the difference between the union and the intersection of two regions.

Parameters

SourceA	Specifies the the first of two regions for the computation.
SourceB	Specifies the the second of two regions for the computation.
DestinationRegion	Stores the result of the computation.

Error Code

BadImplementation

Implementation Specifics

This Xlib subroutine is part of AIXwindows Run Time Environment in AIXwindows Environment/6000.

Related Information

The **XIntersectRegion** subroutine, the **XUnionRegion** subroutine.

Enhanced X–Windows Protocols

AllocColor Protocol Request

Purpose

Allocates a read-only color map entry.

Protocol Request Format

Colormap: **COLORMAP**
Red, Green, Blue: **CARD16**
=>
Pixel: **CARD32**
Red, Green, Blue: **CARD16**

Description

The **AllocColor** protocol request allocates a read-only colormap entry corresponding to the closest available RGB values supported by the hardware. The **AllocColor** protocol request returns the pixel and the RGB values actually used.

Fields

Blue	Specifies the blue color value actually used.
Blue	Returns the blue color value actually used.
Colormap	Specifies the colormap.
Green	Specifies the green color value actually used.
Green	Returns the green color value actually used.
Pixel	Returns the number of bitplanes used in a particular window or pixmap.
Red	Specifies the red color value actually used.
Red	Returns the red color value actually used.

Implementation Specifics

This protocol is part of AIXwindows Run Time Environment in AIXwindows Environment/6000.

Related Information

The **XAllocColor** subroutine.

AllocColorCells Protocol Request

Purpose

Allocates color cells.

Protocol Format

Colormap: **COLORMAP**
Colors, *Planes*: **CARD16**
Contiguous: **BOOL**

=>

Pixels, *Masks*: LISTofCARD32

Description

The **AllocColorCells** protocol request allocates color cells. The number of colors must be positive and the number of planes must be non-negative or a **Value** error results. This protocol request 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 together, `C*2**P` distinct pixels values can be produced. These pixel values are allocated writable by the protocol request .

If *Contiguous* is a value of **True** and all masks are ORed together, the following sets of bits are formed:

- A single contiguous set of bits for the **GrayScale** or **PseudoColor** colormap class (each mask has exactly one bit set to 1).
- Three contiguous sets of bits (one within each pixel subfield) for the **DirectColor** colormap class (each mask has exactly three bits set to 1).

A **Value** error occurs if the number of colors is non-positive or the number of planes is negative.

Fields

Colormap	Specifies the colormap.
Colors	Specifies the number of colors to be allocated.
Planes	Specifies the number of planes to be allocated.
Contiguous	Specifies whether the set of bits formed will be contiguous.
Pixels	Specifies the pixels returned.
Masks	Specifies the masks returned.

Implementation Specifics

This protocol is part of AIXwindows Run Time Environment in AIXwindows Environment/6000.

Related Information

The **XAllocColorCells** subroutine.

AllocColorPlanes Protocol Request

Purpose

Allocates writable color planes.

Protocol Format

Colormap: **COLORMAP**
Colors, *Reds*, *Greens*, *Blues*: **CARD16**
Contiguous: **BOOL**

=>

Pixels: **LISTofCARD32**
RedMask, *GreenMask*, *BlueMask*: **CARD32**

Description

The **AllocColorPlanes** protocol request brings the *Masks* and *Pixels* fields together to produce distinct pixels. The number of colors must be positive and the reds, greens, and blues 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 the R, G, and B bits set respectively.

If the *Contiguous* field is the value of **True**, then each mask will have a contiguous set of bits. No mask will have any bits in common with other masks or with any of the values in the *Pixels* fields.

For the **DirectColor** colormap class, each mask will lie within the corresponding *Pixels* subfield. By ORing together subsets of masks with pixels, `C*2**(R+G+B)` distinct pixels can be produced; these masks are allocated by the protocol.

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 the **PseudoColor** colormap class.

When the colormap entry for a pixel value is changed using the **StoreColors** protocol request or the **StoreNamedColor** protocol request, the *Pixels* field is decomposed according to the masks and the corresponding independent entries are updated.

A **Value** error occurs if the number of colors is negative or the number of reds, greens, and blues is non-positive.

Fields

Colormap	Specifies the colormap.
Colors	Specifies the number of colors to be allocated.
Reds	Specifies the red values used.
Greens	Specifies the green values used.
Blues	Specifies the blue values used.
Contiguous	Specifies whether each mask will have a contiguous set of bits.

Pixel	Specifies the number of pixels returned.
RedMask	Specifies the red mask returned.
GreenMask	Specifies the green mask returned.
BlueMask	Specifies the blue mask returned.

Implementation Specifics

This protocol is part of AIXwindows Run Time Environment in AIXwindows Environment/6000.

Related Information

The **XAllocColorPlanes** subroutine.

AllocNamedColor Protocol Request

Purpose

Searches for the named color of the screen associated with a specified colormap.

Protocol Format

Colormap: **COLORMAP**
Name: **STRING8**
=>
Pixel **CARD32**
ExactRed, ExactGreen, ExactBlue: **CARD16**
VisualRed, VisualGreen, VisualBlue: **CARD16**

Description

The **AllocNamedColor** protocol request searches for the named color of the screen associated with the specified colormap. Then, this protocol completes an **AllocColor** protocol request on the *Colormap* fields. The name should use the ISO Latin-1 encoding. ****The name is not case–sensitive.****

The exact RGB values specify the true values for the color and the visual values specify the values used in the colormap.

Fields

Colormap	Specifies the colormap.
Name	Specifies the named color.
Pixel	Specifies the pixels returned.
ExactRed	Specifies the true red value.
ExactGreen	Specifies the true green value.
ExactBlue	Specifies the true blue value.
VisualRed	Specifies the red value actually used.
VisualGreen	Specifies the green value actually used.
VisualBlue	Specifies the blue value actually used.

Implementation Specifics

This protocol is part of AIXwindows Run Time Environment in AIXwindows Environment/6000.

Related Information

The **XAllocNamedColor** subroutine.

AllowEvents Protocol Request

Purpose

Releases queued events if the client has caused a device to freeze.

Protocol Format

Mode: {**AsyncPointer, SyncPointer, ReplayPointer, AsyncKeyboard, SyncKeyboard, ReplayKeyboard, AsyncBoth, SyncBoth**}
Time: **TIMESTAMP** or **CurrentTime**

Description

The **AllowEvents** protocol request releases some queued events if the client has caused a device to freeze. This protocol has no effect if the specified *Time* field is earlier than the last-grab time of the most recent active grab for the client or if the specified *Time* field 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 the *Mode* field is the **AsyncPointer** mode 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, the **AsyncPointer** mode releases both.

  The **AsyncPointer** mode 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 the *Mode* field is the **SyncPointer** mode 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.

  The **SyncPointer** mode has no effect if the pointer is not frozen by the client or if the pointer is not grabbed by the client.

- If the *Mode* field is the **ReplayPointer** mode 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** protocol request or a previous **AllowEvents** protocol request with the *Mode* field the **SyncPointer** mode but not from a **GrabPointer** protocol request). 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.

  The **ReplayPointer** mode 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 the *Mode* field is the **AsyncKeyboard** mode 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, the **AsyncKeyboard** mode releases both.

  The **AsyncKeyboard** mode 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 the *Mode* field is the **SyncKeyboard** mode, 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, 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.

  The **SyncKeyboard** mode has no effect if the keyboard is not frozen by the client or if the keyboard is not grabbed by the client.

- If the *Mode* parameter is the **ReplayKeyboard** mode and the keyboard is actively grabbed by the client, and is frozen as the result of an event having been sent to the client (either from a **GrabKey** protocol request, or from a previous **AllowEvents** protocol request with the *Mode* field in the **SyncKeyboard** field, but not from a **GrabKeyboard** protocol request); 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.

  The **ReplayKeyboard** mode 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 the *Mode* field is the **SyncBoth** mode, 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. However, if 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 then a subsequent event for it will cause both devices to freeze.

  The **SyncBoth** mode 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, the **SyncBoth** mode releases both (but a subsequent freeze for the **SyncBoth** mode freezes each device only once).

- If the *Mode* field is the **AsyncBoth** mode 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, the **AsyncBoth** mode releases both.

  The **AsyncBoth** mode has no effect unless both pointer and keyboard are frozen by the client.

The **AsyncPointer**, **SyncPointer**, and **Replay Pointer** modes have no effect on processing of keyboard events. The **AsyncKeyboard**, **SyncKeyboard**, and **ReplayKeyboard** modes have no effect on processing of pointer events.

Fields

Mode	Specifies the keyboard and pointer modes.
Time	Specifies the time.

Implementation Specifics

This protocol is part of AIXwindows Run Time Environment in AIXwindows Environment/6000.

Related Information

The **XAllowEvents** subroutine.

Bell Protocol Request

Purpose

Regulates the volume of the keyboard bell.

Protocol Format

Percent: **INT8**

Description

The **Bell** protocol request rings the bell on the keyboard at a volume relative to the base volume for the keyboard, if possible. The *Percent* field can range from –100 to 100 inclusive. The volume at which the bell is rung when the *Percent* field is non-negative will be the following:

base – [(base * percent) / 100] + percent

and when the *Percent* field is negative, will be the following:

base + [(base * percent) / 100]

Fields

Percent Specifies the desired level of volume.

Implementation Specifics

This protocol is part of AIXwindows Run Time Environment in AIXwindows Environment/6000.

Related Information

The **XBell** subroutine.

ChangeActivePointerGrab Protocol Request

Purpose

Changes the specified dynamic fields if the pointer is grabbed by client.

Protocol Format

EventMask: **SETofPOINTEREVENT**
Cursor: **CURSOR** or **None**
Time: **TIMESTAMP** or **CurrentTime**

Description

The **ChangeActivePointerGrab** protocol request changes specified dynamic fields if the pointer is actively grabbed by the client and the specified *Time* field is no earlier than the last-pointer-grab time and no later than the current server time. This protocol request has no effect on the passive fields of a **GrabButton** protocol request.

If a *Cursor* field is specified, it is displayed regardless of which window contains the pointer. If a *Cursor* field is not specified when the pointer is in the *GrabWindow* or one of its subwindows, the normal cursor for that window is displayed. Otherwise, the *Cursor* field for the *GrabWindow* parameter is displayed.

The **ChangeActivePointerGrab** protocol request generates events if the *EventMask* field specified is in the **Set of Event** masks.

Fields

EventMask	Specifies a mask from the **Set of Event** masks.
Cursor	Specifies the cursor.
Time	Specifies the time.

Implementation Specifics

This protocol is part of AIXwindows Run Time Environment in AIXwindows Environment/6000.

Related Information

The **XChangeActivePointerGrab** subroutine.

ChangeGC Protocol Request

Purpose

Changes components in the graphics context.

Protocol Format

GraphicsContext: **GCONTEXT**
ValueMask: **BITMASK**
ValueList: **LISTofVALUE**

Description

The **ChangeGC** protocol request changes components in the *GraphicsContext* field. The *ValueMask* and *ValueList* fields specify which components to be changed. The values and restrictions are the same as for the **CreateGC** protocol request.

Changing the clip-mask overrides any previous **SetClipRectangles** protocol request on the context. Changing the dash-offset or dashes overrides any previous **SetDashes** protocol 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.

Fields

GraphicsContext	Specifies the graphics context.
ValueMask	Specifies which components are to be changed.
ValueList	Specifies which components are to be changed.

Error Codes

Alloc

Font

GContext

Match

Pixmap

Value

Implementation Specifics

This protocol is part of AIXwindows Run Time Environment in AIXwindows Environment/6000.

Related Information

The **CreateGC** protocol request.

The **XChangeGC** subroutine, **XSetArcMode** subroutine, **XSetArcMode** subroutine, **XSetBackground** subroutine, **XSetClipMask** subroutine, **XSetClipOrigin** subroutine, **XSetFillRule** subroutine, **XSetFillStyle** subroutine, **XSetFont** subroutine, **XSetForeground** subroutine, **XSetFunction** subroutine, **XSetGraphicsExposures** subroutine, **XSetLineAttributes** subroutine, the**XSetPlaneMask** subroutine, **XSetState** subroutine,

XSetStipple subroutine, **XSetSubwindowMode** subroutine, **XSetTile** subroutine, **XSetTSOrigin** subroutine.

ChangeHosts Protocol Request

Purpose

Adds or removes the specified host from the access control list.

Protocol Format

Mode: **{Insert, Delete}**
Host: **HOST**

Description

The **ChangeHosts** protocol request 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 must reside on the same host as the server or the client must have permission by a server-dependent method to run this protocol 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 can support families that are not defined.

For the Internet family, the address must be 4 bytes long. The address bytes are in standard IP order. The server performs no automatic swapping on the address bytes.

- For the Class A address, the network number is the first byte in the address, and the host number is the remaining 3 bytes with the most-significant byte first.
- For the Class B address, the network number is the first 2 bytes and the host number is the last 2 bytes with the most-significant byte first.
- For the Class C address, the network number is the first 3 bytes with the most-significant byte first. The last byte is the host number.

For the DECnet family, the server performs no automatic swapping on the address bytes. A Phase IV address is 2 bytes long: the first byte contains the least-significant eight bits of the node number, and the second byte contains the most-significant two bits of the node number in the least-significant two bits of the byte and the area in the most significant six bits of the byte.

For the Chaos family, the address must be 2 bytes long. The host number is always the first byte in the address, and the subnet number is always the second byte. The server performs no automatic swapping on the address bytes.

Use of an unsupported family or an improper address format or length within a supported family results in a **Value** error.

Fields

Mode	Specifies the mode.
Host	Specifies the host.

Implementation Specifics

This protocol is part of AIXwindows Run Time Environment in AIXwindows Environment/6000.

Related Information

The **XAddHost** subroutine, **XAddHosts** subroutine, **XRemoveHost** subroutine, **XRemoveHosts** subroutine.

ChangeKeyboardControl Protocol Request

Purpose

Controls various aspects of the keyboard.

Protocol Format

ValueMask: **BITMASK**
ValueList: **LISTofVALUE**

Description

The **ChangeKeyboardControl** protocol request controls various aspects of the keyboard. The *ValueMask* and *ValueList* fields specify which controls are to be changed. The possible values for the *ValueMask* and *ValueList* fields are the following:

Control	Type
AutoRepeatMode:	{On, Off, Default}
BellDuration:	INT16
BellPercent:	INT8
BellPitch:	INT16
Key:	KEYCODE
KeyClickPercent:	INT8
Led:	CARD8
LedMode:	{On, Off}

AutoRepeatMode — If specified with the *Key* value, the *AutoRepeat* mode of only that key is changed if possible. If the *AutoRepeatMode* value only is specified, the global *AutoRepeat* mode for the entire keyboard is changed without affecting the per-key settings.

BellDuration — Sets the duration (in milliseconds) of the bell if possible. To restore the default, set it to a value of –1. Other negative values generate a **Value** error.

BellPercent — Sets the base volume for the bell between 0 (off) and 100 (loud) inclusive if possible. To restore the default, set it to a value of –1. Other negative values generate a **Value** error.

BellPitch — Sets the pitch (in Hz) of the bell. To restore the default, set it to a value of –1. Other negative values generate a **Value** error.

Key — If specified without an *AutoRepeatMode* value , a **Match** error is returned.

KeyClickPercent — Sets the volume for key clicks between 0 (off) and 100 (loud) inclusive. To restore the default, set it to a value of –1. Other negative values generate a **Value** error.

No standard interpretation of the LEDs is defined. Numbering from one, 32 LEDs are supported.

- If both the *LedMode* and the *Led* are specified, the state of that LED is changed.
- If only the *LedMode* is specified, the state of all LEDs is changed, if possible.
- If an LED is specified without the *LedMode*, a **Match** error is returned.

Each key has an individual mode of whether it should auto-repeat and a default setting for that mode. In addition, there is a global mode of whether auto-repeat should be enabled and a default setting for that mode. When the global mode is set to **On**, keys should obey their individual auto-repeat modes. When the global mode is set to **Off**, no keys should auto-repeat. An auto-repeating key generates alternating **KeyPress** and **KeyRelease** events. When a key is used as a modifier, it is desirable for the key not to auto-repeat, regardless of the auto-repeat setting for that key. When a key is specified without an auto-repeat mode, a **Match** error is generated.

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.

Fields

ValueMask	Specifies the arguments to be provided.
ValueList	Contains one value for each bit set to 1 in the mask.

Implementation Specifics

This protocol is part of AIXwindows Run Time Environment in AIXwindows Environment/6000.

Related Information

The **XAutoRepeatOff** subroutine, **XAutoRepeatOn** subroutine, **XChangeKeyboardControl** subroutine.

ChangeKeyboardMapping Protocol Request

Purpose

Defines the symbols for the specified number of key codes.

Protocol Format

FirstKeycode: **KEYCODE**
KeysymsPerKeycode: **CARD8**
Keysyms: **LISTofKEYSYM**

Description

The **ChangeKeyboardMapping** protocol request defines the symbols for the specified number of key codes, starting with the specified key code. The symbols for key codes outside this range remain unchanged. This protocol request generates a **MappingNotify** event.

The number of elements in the *Keysyms* field list must be a multiple of the *KeysymsPerKeycode* field, otherwise a **Length** error is returned.

The *FirstKeycode* field must be greater than or equal to the *MinKeycode* field as returned in the connection setup, and

```
FirstKeycode + (Keysyms_length/KeysymsPerKeycode) -1
```

must be less than or equal to the *MaxKeycode* field as returned in the connection setup or a **Value** error results. The KEYSYM number *N* (counting from 0) for the *Keycode* K field has an index (counting from 0) of

```
(K - FirstKeycode) * KeysymsPerKeycode + N
```

in the *Keysyms* field.

The *KeysymsPerKeycode* field value can be chosen arbitrarily by the client to be large enough to hold the necessary symbols. A special KEYSYM value of the **NoSymbol** value should be used to fill in unused elements for individual key codes. The **NoSymbol** value can be used in non-trailing positions of the effective list for a key code.

The server does not have to interpret this mapping; it merely stores it for reading and writing by clients.

Fields

FirstKeycode	Specifies the first physical key.
KeysymsPerKeycode	Specifies the mapping of key symbols to key codes.
Keysyms	Specifies the encoding of a symbol on the cap of a key.

Implementation Specifics

This protocol is part of AIXwindows Run Time Environment in AIXwindows Environment/6000.

Related Information

The **XChangeKeyboardMapping** subroutine.

ChangePointerControl Protocol Request

Purpose

Defines how the pointer moves.

Protocol Format

DoAcceleration, DoThreshold: **BOOL**
AccelerationNumerator, AccelerationDenominator: **INT16**
Threshold: **INT16**

Description

The **ChangePointerControl** protocol request 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 the *Threshold* field pixels at once and applies to the amount beyond the *Threshold* field only.

To restore the default, set to a value of –1. Other negative values generate a **Value** error. A value of 0 for the *AccelerationDenominator* field also generates a **Value** error.

Fields

DoAcceleration	Specifies a Boolean value that controls whether the values for the *AccelerationNumerator* and *AccelerationDenominator* fields are used.
DoThreshold	Specifies a Boolean value that controls whether the value for the *Threshold* field is used.
AccelerationNumerator	Specifies the pointer movement speed multiplier.
AccelerationDenominator	Specifies the normal pointer movement speed.
Threshold	Specifies the acceleration threshold, in pixels.

Implementation Specifics

This protocol is part of AIXwindows Run Time Environment in AIXwindows Environment/6000.

Related Information

The **XChangePointerControl** subroutine.

ChangeProperty Protocol Request

Purpose

Alters the property for a specified window.

Protocol Format

Window: **WINDOW**
Property, *Type*: **ATOM**
Format: **{8, 16, 32}**
Mode: **{Replace, Prepend, Append}**
Data: **LISTofINT8** or **LISTofINT16** or **LISTofINT32**

Description

The **ChangeProperty** protocol request alters the property for the specified window. This protocol request generates a **PropertyNotify** event on the window.

The maximum size of a property is server-dependent and may vary dynamically. 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.

Fields

Window — Specifies the window.

Property — Defines the property of the window. If the *Property* field is undefined, it is treated as defined with the correct *Type* and *Format* fields with data that has the length of 0.

Type — Specifies an arbitrary atom used to identify the interpretation of property data. The *Type* field is uninterpreted by the server.

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.

Mode — Specifies what to do with the data. If the *Mode* field is the following:

Replace — The previous *Property* field value is discarded.

Prepend — Adds the data to the beginning of the existing data.

Append — Adds the data to the end of the existing data.

For the **Prepend** or **Append** value, the *Type* and *Format* fields must match the existing value of the *Property* field or a **Match** error results.

Data — Specifies the description of the properties.

Implementation Specifics

This protocol is part of AIXwindows Run Time Environment in AIXwindows Environment/6000.

Related Information

The **XChangeProperty** subroutine, **XSetCommand** subroutine, **XSetIconSizes** subroutine, **XSetNormalHints** subroutine, **XSetSizeHints** subroutine, **XSetStandardProperties** subroutine, **XSetWMHints** subroutine, **XSetZoomHints** subroutine, **XStoreBuffer** subroutine, **XStoreBytes** subroutine, **XStoreName** subroutine.

ChangeSaveSet Protocol Request

Purpose

Adds or removes the specified window from the client save-set.

Request Format

Window: **WINDOW**
Mode: {**Insert, Delete**}

Description

The **ChangeSaveSet** protocol request adds or removes the specified window from the client save-set. The specified window must be created by another client or a **Match** error results. The server automatically removes windows from the save-set when the client is destroyed.

Fields

Window	Specifies the window.
Mode	Specifies how to deal with specified window.

Implementation Specifics

This protocol is part of AIXwindows Run Time Environment in AIXwindows Environment/6000.

Related Information

Closing the Connections to the Server, Save Set

The **XAddToSaveSet** subroutine. **XChangeSaveSet** subroutine, **XRemoveFromSaveSet** subroutine.

ChangeWindowAttributes Protocol Request

Purpose

Changes window attributes.

Protocol Format

Window: **WINDOW**
ValueMask: **BITMASK**
ValueList: **LISTofVALUE**

Description

The **ChangeWindowAttributes** protocol request changes window attributes. The *ValueMask* and *ValueList* fields specify which attributes to change. The values and restrictions are the same as for the **CreateWindow** protocol request, as follows:

- Setting a new background, using either the background pixmap or background pixel, overrides any previous background.
- Setting a new border with the border pixel or border pixmap overrides any previous border.
- Changing the background does not cause the window contents to be changed.
- Setting the border or changing the background so that the border tile origin changes, causes the border to be repainted.
- Changing the background of a root window to the value of **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 but their 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**

  An attempt to violate these restrictions results in an **Access** error.
- There is only one do_not_propagate_mask per window, not one per client.
- Changing the *Colormap* attribute of a window, by defining a new colormap, not by changing the contents of the existing map, generates a **ColormapNotify** event.

- Changing the *Colormap* attribute of a visible window may have no immediate effect on the screen.
- Changing the *Cursor* attribute of a root window to the value of **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 may have been altered.

Fields

Window	Specifies the window.
ValueMask	Specifies which components are to be changed.
ValueList	Specifies which components are to be changed.

Implementation Specifics

This protocol is part of AIXwindows Run Time Environment in AIXwindows Environment/6000.

Related Information

The **CreateWindow** protocol request, **InstallColormap** protocol request.

The **XChangeWindowAttributes** subroutine, **XDefineCursor** subroutine, **XSelectInput** subroutine, **XUndefineCursor** subroutine, **XSetWindowBackground** subroutine, **XSetWindowBackgroundPixmap** subroutine, **XSetWindowBorder** subroutine, **XSetWindowBorderPixmap** subroutine, **XSetWindowColormap** subroutine.

CirculateWindow Protocol Request

Purpose

Circulates the specified window in a specified direction.

Protocol Format

Window: **WINDOW**
Direction: **{RaiseLowest, LowerHighest}**

Description

The **CirculateWindow** protocol request circulates the specified window in the specified direction.

The **CirculateWindow** protocol request generates a **CirculateNotify** event if the window is restacked. This protocol request generates a **CirculateRequest** event if another client selects the **SubstructureRedirect** event on the window. In this case, no further processing is performed.

Exposure processing is performed on formerly obscured windows.

Fields

Window Specifies the window.

Direction Specifies the direction to circulate the specified window. The *Direction* field can be the following:

RaiseLowest It raises the lowest mapped child window, if any, that is occluded by another child window, to the top of the stack.

LowerHighest It lowers the highest mapped child window, if any, that occludes another child window, to the bottom of the stack.

Implementation Specifics

This protocol is part of AIXwindows Run Time Environment in AIXwindows Environment/6000.

Related Information

The **XCirculateSubwindows** subroutine, **XCirculateSubwindowsDown** subroutine, **XCirculateSubwindowsUp** subroutine.

ClearArea Protocol Request

Purpose

Clears the area within a window.

Protocol Format

Window: **WINDOW**
X, Y: **INT16**
Width, *Height*: **CARD16**
Exposures: **BOOL**

Description

The **ClearArea** protocol request clears the area within a window.

In the following cases, if the *Exposures* field is the value of **True**, one or more exposure events for regions of the rectangle that are visible or in a backing store are generated.

- If the *Window* field has a defined background tile, the rectangle is tiled with a plane mask of all ones and has a **Copy** function and has a subwindow-mode of **ClipByChildren**.
- If the *Window* field has a background the value of **None**, the contents of the window are not changed.

If the **ClearArea** protocol request is used with an **InputOnly** window, a **Match** error is generated.

Fields

Window	Specifies the window.
X	Specifies the x coordinate of the upper-left corner of the rectangle relative to the origin of the specified window.
Y	Specifies the y coordinate of the upper-left corner of the rectangle relative to the origin of the specified window.
Width	Specifies the width of the specified window. If the *Width* field is a value of 0, it is replaced with the current width of the window minus x.
Height	Specifies the height of the specified window. If the *Height* field is a value of 0, it is replaced with the current height of the window minus y.
Exposures	Indicates whether exposure events are generated for regions of the rectangle that are either visible or are being retained in a backing store.

Implementation Specifics

This protocol is part of AIXwindows Run Time Environment in AIXwindows Environment/6000.

Related Information

The **XClearArea** subroutine, **XClearWindow** subroutine.

CloseFont Protocol Request

Purpose

Deletes the association between the resource ID and the font.

Protocol Format

Font: **FONT**

Description

The **CloseFont** protocol request deletes the association between the resource ID and the font. The font is freed when no other resource references it.

Field

Font	Specifies the font.

Implementation Specifics

This protocol is part of AIXwindows Run Time Environment in AIXwindows Environment/6000.

Related Information

The **XFreeFront** subroutine, **XUnloadFont** subroutine.

ConfigureWindow Protocol Request

Purpose

Reconfigures the size, position, border, and stacking order of a window.

Protocol Format

Window: **WINDOW**
ValueMask: **BITMASK**
ValueList: **LISTofVALUE**

Description

The **ConfigureWindow** protocol request 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 the *ValueMask* and *ValueList* fields. The values not specified are taken from the existing geometry of the window. Possible values for the *ValueList* field are the following:

X:	**INT16**
Y:	**INT16**
W*idth*:	**CARD16**
Height:	**CARD16**
BorderWidth:	**CARD16**
Sibling:	**WINDOW**
StackMode:	{**Above, Below, TopIf, BottomIf, Opposite**}

- The *X* and *Y* attributes, which are coordinates relative to the origin of the parent window, specify the position of the upper-left outer corner of the window.
- The *Width* and *Height* attributes specify the inside of the window, excluding the border. The width and height must be a nonzero value, or a **Value** error results.
- Changing just the *BorderWidth* attribute 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 *BorderWidth* field of an **InputOnly** window is a nonzero, a **Match** error is generated.

  If the *OverrideRedirect* attribute of the window has the value of **False** and another client has selected the **SubstructureRedirect** event on the parent, then a **ConfigureRequest** event is generated, and no further processing is performed.

  If another client selects the **ResizeRedirect** event on the window and the inside width or height of the window is being changed, then a the **ResizeRequest** event is generated and the current inside width and height are used instead.

  The *OverrideRedirect* attribute of the window has no effect on the **ResizeRedirect** event, and the **SubstructureRedirect** event on the parent window has precedence over the **ResizeRedirect** event on the window.

  If the window state changes, the geometry of the window is changed as specified, the window is restacked among siblings, and a **ConfigureNotify** event is generated.

- If the inside *Width* or *Height* field of the window is changed, the children of the window are affected according to window gravity.
  - 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 the 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 window gravity.

  For a change of width *W* and height *H* the following values are defined as [*X, Y*] pairs:

NorthWest	[0,0]
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 window 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 the *BitGravity* attribute, the change in position of each pixel is [*–X, –Y*], and for the *WindowGravity* attribute, the change in position of a child when its parent is resized is [*–X, –Y*]. The **Static** gravity takes effect only when the width or height of the window is changed, not when the window is moved only.

- A *BitGravity* attribute of **Forget** indicates the following (even if backing store or save-under has been requested):
  - The window contents are always discarded after a size change
  - The window is tiled with its background. If no background is defined, the existing screen contents are not altered.
  - Zero or more exposure events are generated.

  A server can ignore the specified *BitGravity* field and use **Forget** instead.
- A *WindowGravity* attribute of **Unmap** is like a **NorthWest** value, 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.
- If *Sibling* and *StackMode* 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.
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 *StackMode* 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 *StackMode* or if the window is not actually a sibling, a **Match** error occurs.

The computations for the **BottomIf**, **TopIf**, and **Opposite** stack modes are performed with respect to the final geometry of the window (as controlled by the other arguments to the protocol), not by the initial geometry of the window.

Fields

Window	Specifies the window.
ValueMask	Specifies which components are to be changed.
ValueList	Specifies which components are to be changed.

Implementation Specifics

This protocol is part of AIXwindows Run Time Environment in AIXwindows Environment/6000.

Related Information

The **XConfigureWindow** subroutine, **XLowerWindow** subroutine, **XMapRaised** subroutine, **XMoveResizeWindow** subroutine, **XMoveWindow** subroutine, **XRaiseWindow** subroutine, **XResizeWindow** subroutine, **XRestackWindows** subroutine, **XSetWindowBorderWidth** subroutine.

ConvertSelection Protocol Request

Purpose

Converts a selection.

Protocol Format

Selection, Target: **ATOM**
Property: **ATOM** or **None**
Requestor: **WINDOW**
Time: **TIMESTAMP** or **CurrentTime**

Description

The **ConvertSelection** protocol request converts a selection by sending a **SelectionRequest** event to its owner. The fields remain unchanged from their original state. This protocol does not change the property.

If the specified *Selection* field has an owner, the server sends a **SelectionRequest** event to that owner. If there is no owner for the specified *Selection* field, the server generates a **SelectionNotify** event to the requester with the property specified as the **None** value.

Fields

Selection	Specifies what is to be converted.
Target	Specifies the type of the selection to be converted.
Property	Specifies the property of the window.
Requestor	Specifies the client making the request.
Time	Specifies the time in a timestamp or the **CurrentTime**.

Implementation Specifics

This protocol is part of AIXwindows Run Time Environment in AIXwindows Environment/6000.

Related Information

The **XConvertSelection** subroutine.

CopyArea Protocol Request

Purpose

Combines the specified source and destination rectangles.

Protocol Format

SourceDrawable, DestinationDrawable: **DRAWABLE**
GraphicsContext: **GCONTEXT**
SourceX, SourceY: **INT16**
Width, Height: **CARD16**
DestinationX, DestinationY: **INT16**

Description

The **CopyArea** protocol request combines the specified rectangle of the source drawable with the specified rectangle of the destination drawable. The *SourceX* and *SoucreY* fields are relative to the source drawable's origin. The *DestinationX* and *DestinationY* fields are relative to the destination drawable's origin, each pair specifying the upper-left corner of the rectangle. The source drawable and the destination drawable must have the same root and the same depth, or a **Match** error occurs.

If regions of the source rectangle are obscured and have not been retained in backing-store or if regions outside the boundaries of the source drawable are specified, those regions are not copied. However, the following occurs on all corresponding destination regions that are either visible or retained in backing-store. If the destination drawable is a window with a background other than the value of **None**, these corresponding destination regions are tiled (with plane-mask of all ones and function **Copy**) with that background.

Regardless of tiling and whether the destination is a window or a pixmap, if graphics-exposures in *GraphicsContext* field is the value of **True**, **GraphicsExpose** events for all corresponding destination regions are generated. If graphics-exposure is the value of **True** but no **GraphicsExpose** events are generated, a **NoExpose** event is generated.

Fields

DestinationDrawable	Specifies the destination drawable of the specified rectangle.
DestinationX	Specifies the x coordinate of the upper-left corner of the rectangle relative to the origin of the destination drawable.
DestinationY	Specifies the y coordinate of the upper-left corner of the rectangle relative to the origin of the destination drawable
GraphicsContext	Specifies the graphics context. This field has the following components: function, plane-mask, subwindow-mode, graphics-exposures, clip-x-origin, clip-y-origin, and clip-mask.
Height	Specifies the height of the specified rectangle.
SourceDrawable	Specifies the source drawable of the specified rectangle.
SourceX	Specifies the x coordinate of the upper-left corner of the rectangle relative to the origin of the source drawable.

SourceY	Specifies the y coordinate of the upper-left corner of the rectangle relative to the origin of the source drawable.
Width	Specifies the width of the specified rectangle.

Implementation Specifics

This protocol is part of AIXwindows Run Time Environment in AIXwindows Environment/6000.

Related Information

The **XCopyArea** subroutine.

CopyColormapAndFree Protocol Request

Purpose

Creates a colormap.

Protocol Format

MapID, SourceColormap: **COLORMAP**

Description

The **CopyColormapAndFree** protocol request creates a colormap of the same visual type and for the same screen as specified in the *SourceColormap* field, and associates the map identifier with it.

The **CopyColormapAndFree** protocol request moves all of the existing allocations of the client from the *SourceColormap* field to the new colormap with the color values intact, and their read-only or writable characteristics intact. This protocol also frees these colormap entries in the *SourceColormap* field. Color values in other entries in the new colormap are undefined.

If the *SourceColormap* field was created by the client with the *Allocate* attribute set to the value of **All** in the **CreateColormap** protocol request, then the following occurs:

- A new colormap is also created with the *Allocate* attribute set to the value of **All**.
- The color values for all entries are copied from the *SourceColormap* field.
- All entries in the *SourceColormap* field are freed.

If the *SourceColormap* was not created by the client with the *Allocate* attribute set to the value of **All**, the allocations to be moved are all pixels and planes that have been allocated by the client using **AllocColor**, **AllocColorCells**, **AllocColorPlanes**, or **AllocNamedColor**, and that have not been freed since they were allocated.

Fields

MapID	Specifies the identifier for the source colormap.
SourceColormap	Specifies the source colormap.

Implementation Specifics

This protocol is part of AIXwindows Run Time Environment in AIXwindows Environment/6000.

Related Information

The **CreateColormap** protocol.

The **XCopyColormapAndFree** subroutine.

CopyGC Protocol Request

Purpose

Copies graphics context (**GC**) components from a source to a destination.

Protocol Format

SourceGraphicsContext, DestinationGraphicsContext: **GCONTEXT**
ValueMask: **BITMASK**

Description

The **CopyGC** protocol request copies components from the *SourceGraphicsContext* field to the *DestinationGraphicsContext* field. The source **GC** and the destination **GC** must have the same root and the same depth or a **Match** error results.

Fields

SourceGraphicsContext	Specifies the **GC** components to be copied.
DestinationGraphicsContext	Specifies the location the **GC** components will be copied to.
ValueMask	Specifies which components to copy. The **CreateGC** protocol request lists these components.

Implementation Specifics

This protocol is part of AIXwindows Run Time Environment in AIXwindows Environment/6000.

Related Information

The **CreateGC** protocol.

The **XCopyGC** subroutine.

CopyPlane Protocol Request

Purpose

Combines the source drawable with the destination drawable.

Protocol Format

SourceDrawable, DestinationDrawable: **DRAWABLE**
GraphicsContext: **GCONTEXT**
SourceX, SourceY: **INT16**
Width, Height: **CARD16**
DestinationX, DestinationY: **INT16**
BitPlane: **CARD32**

Description

The **CopyPlane** protocol request combines the *SourceDrawable* field with the *DestinationDrawable* field. The source drawable and the destination drawable must have the same root or a **Match** error will result. The source and the destination drawables need not have the same depth.

The *BitPlane* field must have exactly one bit set to the value of 1 and the value of *BitPlane* field must be less than 2^{**n}, where *n* is the depth of the source drawable, or a **Value** error results. Effectively, a pixmap of the same depth as the destination drawable with the size specified by the source region is formed using the foreground and background pixels in the graphics context.

- The *BitPlane* in the source drawable contains a 1 bit for foreground.
- The *BitPlane* in the source drawable contains a 0 bit for background.

The equivalent of a **CopyArea** protocol request is performed, with all the same exposure semantics.

The **CopyPlane** protocol request 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.

The **GraphicsContext** components are *function, plane_mask, foreground, background, subwindow_mode, graphics_exposures, clip_x_origin, clip_y_origin,* and *clip_mask.*

Fields

BitPlane	Specifies the bit plane of the source drawable.
DestinationDrawable	Specifies the destination drawable.
DestinationX	Specifies the x coordinate relative to the origin of the destination drawable.
DestinationY	Specifies the y coordinate relative to the origin of the destination drawable
GraphicsContext	Specifies the graphics context.
Height	Specifies the height of the specified plane.

SourceDrawable	Specifies the source drawable of the specified plane.
SourceX	Specifies the x coordinate relative to the origin of the source drawable.
SourceY	Specifies the y coordinate relative to the origin of the source drawable.
Width	Specifies the width of the specified plane.

Implementation Specifics

This protocol is part of AIXwindows Run Time Environment in AIXwindows Environment/6000.

Related Information

The **CopyArea** protocol request.

The **XCopyPlane** subroutine.

CreateColormap Protocol Request

Purpose

Creates a colormap.

Protocol Format

MapID: **COLORMAP**
Visual: **VISUALID**
Window: **WINDOW**
Allocate: {**None, All**}

Description

The **CreateColormap** protocol request creates a colormap of the specified visual type for the screen on which the *Window* field resides, and associates the map identifier with it. The visual type must be supported by the screen, or a **Match** error results.

The initial values of the colormap entries are undefined for **GrayScale**, **PseudoColor**, and **DirectColor**. For these classes, if the value of *Allocate* is set to the value of **None**, clients can allocate the colormap entries.

The colormap entries for **StaticGray**, **StaticColor**, and **TrueColor** are defined values specific to *Visual* field, but not defined by the Core protocol request. *Allocate* field for these classes must be specified as the value of **None**, or a **Match** error results. For the other classes, if the value of *Allocate* field is set to the value of **None**, the colormap initially has no allocated entries, and clients can allocate entries.

If *Allocate* field is set to the value of **All**, then the entire colormap is allocated writable. The initial value of allocated entries is undefined.

- For the **GrayScale** and **PseudoColor** visual types, the effect is as if an **AllocColor** protocol request returned all pixel values from 0 to *N*-1, where *N* is the colormap-entries value in the specified visual.
- For **DirectColor**, the effect is as if an **AllocColorPlanes** protocol request returned a pixel value of 0 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 the **FreeColors** protocol request.

Fields

Allocate	Specifies the colormap entries to be allocated. This field can have the values of **All** or **None**.
MapID	Specifies the identifier for the colormap.
Visual	Specifies the visual ID of the display.
Window	Specifies the window.

Implementation Specifics

This protocol is part of AIXwindows Run Time Environment in AIXwindows Environment/6000.

Related Information

The **XCreateColormap** subroutine, **XResetScreenSaver** subroutine.

CreateCursor Protocol Request

Purpose

Creates a cursor.

Protocol Format

CursorID: **CURSOR**
Source: **PIXMAP**
Mask: **PIXMAP** or **None**
ForegroundRed, *ForegroundGreen*, *ForegroundBlue*: **CARD16**
BackgroundRed, *BackgroundGreen*, *BackgroundBlue*: **CARD16**
X, *Y*: **CARD16**

Description

The **CreateCursor** protocol request creates a cursor to be associated with the *CursorID* field. 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 1 bit in the source.
- The background is used for the 0 bit in the source.

The source pixmap and the mask pixmap must have depth of one (or a **Match** error results). Source and mask pixmaps can have any root. The mask pixmap does not need to be specified.

The *Mask* field pixmap defines the shape of the cursor, or the one bit in the *Mask* field define which pixels in the *Source* field will be displayed.

- If the *Mask* field has 0 bit, the corresponding bits of the *Source* field pixmap are ignored.
- If no *Mask* field is given, all pixels of the *Source* field are displayed.

The *Mask* field and the *Source* field must be the same size or a **Match** error results.

The *X* and *Y* coordinates define the hotspot. These coordinates are relative to the origin of the *Source* field pixmap and must be a point within the *Source* field pixmap or a **Match** error results.

The components of the cursor can be transformed arbitrarily to meet display limitations. The pixmaps can be freed immediately if no further explicit references are 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.

Fields

BackgroundBlue	Specifies the intensity value for the background blue.
BackgroundGreen	Specifies the intensity value for the background green.
BackgroundRed	Specifies the intensity value for the background red.
CursorID	Specifies the identifier for the cursor.
Mask	Defines the shape of the cursor.

Source	Specifies the orgin of the specified cursor.
ForegroundBlue	Specifies the intensity value for the foreground blue.
ForegroundGreen	Specifies the intensity value for the foreground green.
ForegroundRed	Specifies the intensity value for the foreground red.
X	Specifies the x coordinate of the hotspot relative to the orgin of the source.
Y	Specifies the y coordinate hotspot relative to the orgin of the source.

Implementation Specifics

This protocol is part of AIXwindows Run Time Environment in AIXwindows Environment/6000.

Related Information

The **XCreatePixmapCursor** subroutine.

CreateGC Protocol Request

Purpose

Creates a graphics context (**GC**) and assigns an identifier to it.

Protocol Format

GraphicsContextID: **GCONTEXT**
Drawable: **DRAWABLE**
ValueMask: **BITMASK**
ValueList: **LISTofVALUE**

Description

The **CreateGC** protocol request creates a graphics context. This protocol request assigns the identifier as specified in the *GraphicsContextId* field to the newly created **GC**. The **GC** can be used with any destination drawable with the same root and depth as the specified *Drawable* field. Other drawables result in a **Value** error.

The *ValueMask* field and the *ValueList* field specify which components are to be explicitly initialized. The **GC** components include the following:

Component	Type	Default
Function	**{Clear, And, AndReverse, Copy, AndInverted, NoOp, Xor, Or, Nor, Equiv, Invert, OrReverse, CopyInverted, OrInverted, Nand, Set}**	**Copy**
PlaneMask	**CARD32**	all ones
Foreground	**CARD32**	0
Background	**CARD32**	1
LineWidth	**CARD16**	0
LineStyle	**{Solid, OnOffDash, DoubleDash}**	**Solid**
CapStyle	**{NotLast, Butt, Round, Projecting}**	**Butt**
JoinStyle	**{Miter, Round, Bevel}**	**Miter**
FillStyle	**{Solid, Tiled, OpaqueStippled, Stippled}**	**Solid**
FillRule	**{EvenOdd, Winding}**	**EvenOdd**
ArcMode	**{Chord, PieSlice}**	**PieSlice**
Tile	**PIXMAP**	Pixmap of unspecified size filled with *Foreground* pixel (client–specified pixel, if any, otherwise zero). Subsequent changes to *Foreground* pixel do not affect this pixmap.

Stipple	**PIXMAP**	Pixmap of unspecified size filled with ones.
TileStippleXOrigin	**INT16**	0
TileStippleYOrigin	**INT16**	0
Font	**FONT**	<server-dependent-font>
SubwindowMode	**{ClipByChildren, IncludeInferiors}**	**ClipByChildren**
GraphicsExposures	**BOOL**	**True**
ClipXOrigin	**INT16**	0
ClipYOrigin	**INT16**	0
ClipMask	**PIXMAP** or **None**	**None**
DashOffset	**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 *PlaneMask* field restricts the operation to a subset of planes. The result is the following:

```
((source FUNC destination) AND PlaneMask) OR (destination AND (NOT PlaneMask))
```

Range checking is not performed on the values for the *Foreground*, *Background*, or *PlaneMask* fields. These values are truncated to the appropriate number of bits. These field values are defined as the following:

Function	**Operation**
Clear	0
And	source AND destination
AndReverse	source AND (NOT destination)
Copy	source
AndInverted	(NOT source) AND destination
NoOp	destination
Xor	source XOR destination
Or	source OR destination
Nor	(NOT source) AND (NOT destination)
Equiv	(NOT source) XOR destination
Invert	NOT destination
OrReverse	source OR (NOT destination)
CopyInverted	NOT source

OrInverted (NOT source) OR destination

Nand (NOT source) OR (NOT destination)

Set 1

The *LineWidth* field, measured in pixels, can be either of the following:

- A wide line (greater than or equal to one).

  Wide lines are drawn centered on the path described by the graphics protocol request. Unless otherwise specified by the *JoinStyle* or *CapStyle* fields, the bounding box of a wide line with endpoints [x1, y1], [x2, y2], and 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, which is viewed as having infinitely thin edges. 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 (the x increasing direction).

  Pixels with centers on a horizontal edge are part of the line, if and only if, the interior is immediately below (the y increasing direction) and the interior of the boundary is immediately to the right (the x increasing direction).

  NOTE: This description is a mathematical model describing the pixels that are drawn for a wide line and does not imply that trigonometry is required to implement such a model.

  Real-point or fixed-point arithmetic is recommended for computing the corners of the line endpoints for lines greater than one pixel in width.

- A thin line (zero-width lines).

  Thin lines are drawn, one-pixel wide lines that use an unspecified, device-dependent algorithm. The two constraints on this algorithm are the following:

  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 including cap and join styles. It is encouraged that you make this property true for thin lines.

A *LineWidth* field with the value of 0 may differ from a *LineWidth* field with the value of 1 in which pixels are drawn. In general, drawing a thin line is 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 *LineWidth* field with the value of 1, rather than a *LineWidth* field with the value of 0.

The *LineStyle* field defines which sections of a line are drawn:

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 (as in the *FillStyle* field). The **Butt** value 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 the **NotLast** value is treated as the *CapStyle* field value of the **Butt** value).

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

NotLast	This is the equivalent to the **Butt** value, except that for a *LineWidth* field of 0 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 *LineWidth* field centered on the endpoint; equivalent to the **Butt** value for *LineWidth* field of zero.
Projecting	Square at the end, but the path continues beyond the endpoint for a distance equal to half the line width; equivalent to the **Butt** value for a *LineWidth* field of zero.

The *JoinStyle* value defines how corners are drawn for wide lines:

Miter	The outer edges of the two lines extend to meet at an angle.
Round	A circular arc with diameter equal to the *LineWidth* value, centered on the joinpoint.
Bevel	The **Butt** value endpoint styles and then the triangular notchfilled.

For a line with coincident endpoints `(x1=x2, y1=y2)`, when the *CapStyle* field is applied to both endpoints, the semantics depends on the *LineWidth* and *CapStyle* fields.

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 *LineWidth* field.

Projecting wide	The closed path is a square aligned with the coordinate axes centered at the endpoint, with sides equal to the *LineWidth* field.

For a line with coincident endpoints `(x1=x2, y1=y2)`, when the *JoinStyle* field 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 *CapStyle* field is applied at both endpoints.

The *TileStipple* field and clip origins are interpreted relative to the origin of whatever destination drawable is specified in a graphics protocol request.

The *Tile* field pixmap must have the same root and depth as the *GraphicsContext* field or a **Match** error results.

The *Stipple* field pixmap must have depth one and must have the same root as the *GraphicsContext* field or a **Match** error results.

For the *FillStyle* **Stippled** field (but not the *FillStyle* **OpaqueStippled** field), the stipple pattern is tiled in a single plane and acts as an additional clip mask to be ANDed with the *ClipMask* field. Any size pixmap can be used for tiling or stippling, although some sizes can be faster to use than others.

The *FillStyle* field defines the contents of the source for line, text, and fill requests. For all text and fill requests for line requests with the *LineStyle* **Solid** field, and for the even dashes for line requests with a *LineStyle* **OnOffDash** or **DoubleDash** field:

Fill-Style	**Definition**
Solid	Foreground.
Tiled	Tile.
OpaqueStippled	A *Tile* field with the same width and height as stipple, but with background everywhere the *Stipple* field has a value of 0 and with foreground everywhere the *Stipple* field has a value of 1.
Stippled	Foreground masked by the *Stipple* field.

For the odd dashes for line requests with the *LineStyle* **DoubleDash** field:

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* field value is actually a simplified form of the more general patterns that can be set with the **SetDashes** protocol request. Specifying a value of `n` is equivalent to specifying the two-element list `[n, n]` in the **SetDashes** protocol request. The value must be nonzero or a **Value** error results. *DashOffset* and *Dashes* fields are defined in the **SetDashes** protocol request.

The *ClipMask* field restricts writes to the destination *Drawable* field. Only pixels where the *ClipMask* field has a one bit are drawn; pixels are not drawn outside the area covered by the *ClipMask* field or where the *ClipMask* field has a 0 bit. The *ClipMask* field affects all graphics protocol requests, but it does not clip sources. The *ClipMask* field origin is interpreted relative to the origin of whatever destination drawable is specified in a graphics protocol request. If a pixmap is specified as the *ClipMask* field, it must have depth of 1 and have the same root as the **GC**, or a **Match** error results. If the *ClipMask* field is the value of **None**, the pixels are drawn (regardless of the clip origin). The *ClipMask* field also can be set with the **SetClipRectangles** protocol request.

For the **ClipByChildren** type, both source and destination windows are additionally clipped by all viewable **InputOutput** children windows. For the **IncludeInferiors** type, neither source nor destination window is clipped by inferiors; this will result in including subwindow contents in the source and drawing through subwindow boundaries of the destination. The **IncludeInferiors** type can be used on windows with different depths, but the results are undefined by the core protocol request.

The *FillRule* field defines what pixels are inside and are drawn for paths given in the **FillPoly** protocol request as follows.

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 counterclockwise segment is one that 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 *ArcMode* field controls filling in the **PolyFillArc** protocol request.

The *GraphicsExposures* field controls the **GraphicsExposure** event generation for **CopyArea** and **CopyPlane** protocol requests (and any similar protocol requests defined by extensions).

Storing a pixmap in a graphics context might or might not result in a copy being made. If the pixmap is later used as the destination for a graphics protocol request, the change might or might not be reflected in the graphics context. If the pixmap is used simultaneously in a graphics protocol request as both a destination and as a *Tile* or *Stipple* field, the results are not defined.

It is quite likely that some amount of graphics context information will be cached in display hardware and that such hardware can only cache a small number of graphics contexts. Given the number and complexity of components, clients should view switching between graphics contexts with nearly identical states as significantly more time-consuming than making minor changes to a single graphics context.

Fields

GraphicsContextID	Specifies the identifier for the graphics context.
Drawable	Specifies the drawable.
ValueMask	Specifies which graphics context components are to be explicitly initialized.
ValueList	Specifies the values associated with the components in the *ValueMask* field.

Implementation Specifics

This protocol is part of AIXwindows Run Time Environment in AIXwindows Environment/6000.

Related Information

The **XCreateGC** subroutine, **XOpenDisplay** subroutine.

CreateGlyphCursor Protocol Request

Purpose

Creates a cursor and associates an identifier with it.

Protocol Format

CursorID: **CURSOR**
SourceFont: **FONT**
MaskFont: **FONT** or **None**
SourceCharacter, *MaskCharacter*: **CARD16**
ForegroundRed, *ForegroundGreen*, *ForegroundBlue*: **CARD16**
BackgroundRed, *BackgroundGreen*, *BackgroundBlue*: **CARD16**

Description

The **CreateGlyphCursor** protocol request creates a cursor and associates the *CursorID* field with it. This protocol request obtains the source and mask bit maps from the specified font glyphs.

- The *SourceCharacter* field must be a defined glyph in the *SourceFont* field and the *MaskCharacter* field, if it is given, must be a defined glyph in the *MaskFont* field.
- The *MaskFont* and *MaskCharacter* fields are optional.
- The origins of the source and the mask glyphs (if defined) are positioned coincidently and 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 *SourceCharacter* and *MaskCharacter* fields are **CARD16**, not **Char2B**. For 2-byte matrix fonts, the 16-bit value is formed with byte 1 in the most-significant byte and byte 2 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.

Fields

CursorID	Specifies the identifier for the cursor.
SourceFont	Specifies the font for the source glyph.
MaskFont	Specifies the font for the mask glyph.
SourceCharacter	Specifies the character glyph for the source.
MaskCharacter	Specifies the character glyph for the mask.
ForegroundRed	Specifies the intensity value for the foreground red.
ForegroundGreen	Specifies the intensity value for the foreground green.
ForegroundBlue	Specifies the intensity value for the foreground blue.
BackgroundRed	Specifies the intensity value for the background red.
BackgroundGreen	Specifies the intensity value for the background green.

BackgroundBlue	Specifies the intensity value for the background blue.

Implementation Specifics

This protocol is part of AIXwindows Run Time Environment in AIXwindows Environment/6000.

Related Information

The **XCreateFontCursor** subroutine, **XCreateGlyphCursor** subroutine.

CreatePixmap Protocol Request

Purpose

Creates a pixmap and assigns an identifier to it.

Protocol Format

PixmapID: **PIXMAP**
Drawable: **DRAWABLE**
Depth: **CARD8**
Width, *Height*: **CARD16**

Description

The **CreatePixmap** protocol request creates a pixmap and assigns the identifier specified by the *PixmapID* parameter to it.

- The *Width* and *Height* fields must be nonzero or a **Value** error results.
- The *Depth* field must be supported by the root of the specified *Drawable* field or a **Value** error results.
- The initial contents of the pixmap are undefined.

An **InputOnly** window can be used as a *Drawable* field in this protocol request.

Parameters

Depth	Specifies the depth of the pixmap.
Drawable	Specifies the drawable.
Height	Specifies the height of the pixmap.
PixmapID	Specifies the identifier of the pixmap.
Width	Specifies the width of the pixmap.

Implementation Specifics

This protocol is part of AIXwindows Run Time Environment in AIXwindows Environment/6000.

Related Information

The **XCreatePixmap** subroutine.

CreateWindow Protocol Request

Purpose

Creates an unmapped window and assigns an identifier to it.

Protocol Format

WindowID, Parent: **WINDOW**
Class: **{InputOutput, InputOnly, CopyFromParent}**
Depth: **CARD8**
Visual: **VISUALID** or **CopyFromParent**
X, Y: **INT16**
Width, *Height*, *Borderwidth*: **CARD16**
ValueMask: **BITMASK**
ValueList: **LISTofVALUE**

Description

The **CreateWindow** protocol request creates an unmapped window and assigns the *WindowID* field to it. Then, this protocol request generates a **CreateNotify** event.

- If the *Class* field is the **CopyFromParent** value, the class is taken from the parent window.
- A depth of 0 for the **InputOutput** or **CopyFromParent** value will result in the *Depth* field being taken from the parent window.
- If the *Visual* field is the **CopyFromParent** value, the *Visual* field type is taken from the parent window.
- If the *Class* field is the **InputOutput** value, the *Visual* field type and the *Depth* field must be a combination supported for the screen or a **Match** error results. The *Depth* field does not need to be the same as the *Parent* field, but the *Parent* field must not be the **InputOnly** value class or a **Match** error results.
- If the *Class* field is the **InputOnly** value, the *Depth* field must be the value of 0 and the *Visual* field type must be supported for the screen or a **Match** error results; but the *Parent* field can have any *Depth* and *Class* field.

An **InputOnly** window cannot be used for the purposes of graphics requests, exposure processing, and the **VisibilityNotify** events, and cannot be used as a source or destination drawables for graphics protocol requests. Otherwise, the **InputOnly** and **InputOutput** windows act the same with respect to properties, grabs, and input control.

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 window, specify the position of the upper-left outer corner of the window, not the origin.
- The *Width* and *Height* fields specify the inside of the window, excluding the border. The *Width* and *Height* fields must be nonzero or a **Value** error results.
- The *BorderWidth* field for an **InputOnly** window must be a value of 0 or a **Match** error results.
- The *ValueMask* and *ValueList* fields specify window fields that are to be explicitly initialized.

The following are possible values and the default values for the variables in the **CreateWindow** protocol request.

Field	Possible Values	Default Values
BackgroundPixmap	**PIXMAP, None, ParentRelative**	**None**
BackgroundPixel	**CARD32**	No default
BorderPixmap	**PIXMAP** or **CopyFromParent**	**CopyFromParent**
BorderPixel	**CARD32**	No default
BitGravity	**BITGRAVITY**	**Forget**
WinGravity	**WINGRAVITY**	**NorthWest**
BackingStore	{**NotUseful, WhenMapped, Always**}	**NotUseful**
BackingPlanes	**CARD32**	all 1s
BackingPixel	**CARD32**	**Zero**
SaveUnder	**BOOL**	**False**
EventMask	**SETofEVENT**	{} (empty set)
DoNotPropagateMask	**SETofDEVICEEVENT**	{} (empty set)
OverrideRedirect	**BOOL**	**False**
Colormap	**COLORMAP** or **CopyFromParent**	**CopyFromParent**
Cursor	**CURSOR** or **None**	**None**

The window fields in the **CreateWindow** protocol request include the following:

- If the *BackgroundPixmap* field is specified, it overrides the default *BackgroundPixmap*. The *BackgroundPixmap* field and the window must have the same root and the same depth or a **Match** error results. Any size pixmap can be used, although some sizes can be faster than others.
  - If the *BackgroundPixmap* field is the value of **None**, the window has no defined background.
  - If the *BackgroundPixmap* field is the **ParentRelative** value, the background of the parent window is used, but the window must have the same depth as the parent window or a **Match** error results.
  - If the *BackgroundPixmap* field of the parent window is the value of **None**, the window will also have the value of **None** for this variable. A copy of the parent window *BackgroundPixmap* field is not made, but is re-examined each time the window background is required.
- If the *BackgroundPixel* field is specified, it overrides the default the *BackgroundPixmap* field and any of the *BackgroundPixmap* field given explicitly. A pixmap of undefined size filled with *BackgroundPixel* is used for the background. Range checking is not performed on the *BackgroundPixel* field value; it is simply truncated to the appropriate number of bits.
  - With the *BackgroundPixmap* field of the **ParentRelative** value, 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 the *BackgroundPixmap* field with the value of **None**.
- If the *BackgroundPixmap* field is the value of **None**, the previous screen contents from other windows of the same depth as the window are left in place if the contents come from the parent of the window or an inferior of the parent 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 *BackgroundPixmap* field is the value of **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 the *BorderPixmap* field is given, it overrides the default *BorderPixmap* field or a **Match** error results. The *BorderPixmap* 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 the *BorderPixmap* field is the **CopyFromParent** value, the *BorderPixmap* of the parent is copied. Subsequent changes to the parent window border field do not affect the child window. The window must have the same depth as the parent window, or a **Match** error results.

The pixmap can be copied by sharing the same pixmap object between the child and parent window or by making a complete copy of the pixmap contents.

- If the *BorderPixel* field is given, it overrides the default *BorderPixmap* field and any *BorderPixmap* field given explicitly. A pixmap of undefined size filled with *BorderPixel* is used for the border. Range checking is not performed on the *BorderPixel*; it is simply truncated to the appropriate number of bits.
- The *BitGravity* field defines the region of the window that should be retained if the window is resized.
- The *WinGravity* field defines how the window should be repositioned if the parent window is resized. See the **ConfigureWindow** protocol request.
- The *BackingStore* field can be set to one of the following:
  - The **WhenMapped** value, which advises the server that maintaining contents of obscured regions when the window is mapped would be beneficial.
  - The **Always** value, which advises the server that maintaining contents even when the window is unmapped would be beneficial. In this case, the server can generate an exposure event when the window is created. Even if the window is larger than the parent window, the server should maintain complete contents, not just the region within the parent boundaries.
  - The **NotUseful** value, which advises the server that maintaining contents is unnecessary, although a server can still choose to maintain contents while the window is mapped.
  - If the server maintains contents, exposure events will not be generated, but the server may stop maintaining contents at any time. While the server maintains contents, exposure events will not normally be generated, but the server may stop maintaining its contents at any time.

- If the *SaveUnder* field is the value of **True**, it advises the server that when this window is mapped, saving the contents of the windows it obscures would be beneficial.
- When the contents of obscured regions of a window are being maintained, regions obscured by noninferior windows are included in the destination (and source, when the window is the source) of graphics protocol requests, but regions obscured by interior windows are not included.
- The *BackingPlanes* field indicates (with one bits) which bit planes of the window hold dynamic data that must be preserved in the *BackingStores* field and during the *SaveUnders* field.
- The *BackingPixel* field specifies the value to use in planes not covered by *BackingPlanes* field. The server can save only the specified bit planes in the *BackingStore* or *SaveUnder* fields 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 *EventMask* field defines which events the client is interested in for this window or for some event types for inferiors of the window.
- The *DoNotPropagateMask* field defines which events should not be propagated to ancestor windows when no client has the event type selected in this window.
- The *OverrideRedirect* field specifies whether map and configure protocol requests on this window should override a **SubstructureRedirect** event on the parent window. This is typically done to inform the window manager not to tamper with the window.
- The *Colormap* field specifies the colormap that best reflects the true colors of the window. Servers capable of supporting multiple hardware colormaps can use this information, and window managers can use it for **InstallColormap** protocol request. The colormap must have the same visual type as the window or a **Match** error results.

  If the **CopyFromParent** value is specified, the colormap of the parent is copied. However, the window must have the same visual type as the parent window and the parent window must not have a *Colormap* field is set to the value of **None**, or a **Match** error results. Subsequent changes to the parent window do not affect the child window. For an explanation of the value of **None** allocation setting, see the **FreeColormap** protocol request.
- If the *Cursor* field is specified, it will be used whenever the pointer is in the window. If the value of **None** is specified, the cursor of the parent window will be used, and any change in the cursor of the parent window will cause an immediate change in the displayed cursor.

The *BackgroundPixmap*, *BorderPixmap* and the cursor can be freed immediately if no further explicit references to them are made.

Subsequent drawing into the background or *BorderPixmap* has an undefined effect on the window state; the server might or might not make a copy of the pixmap.

The only attributes defined for the **InputOnly** windows are the *WinGravity, EventMask, DoNotPropagateMask, OverrideRedirect*, and *Cursor* fields. Specifying other fields for the **InputOnly** windows generates a **Match** error.

Fields

Parent	Specifies the parent window.
Class	Specifies the class type of the window .
Depth	Specifies the depth of the window.
Visual	Specifies where the visual type of the window.
Width	Specifies the inside width of the window.
Height	Specifies the inside height of the window.
Borderwidth	Specifies the border width of the window.
ValueMask	Specifies the attributes of the window that are to be explicitly initialized.
ValueList	Specifies the values associated with *ValueMask* that are to be explicitly initialized.
WindowId	Specifies the identifier of the window.
X	Specifies the x coordinate that is relative to the origin of the parent window.
Y	Specifies the x coordinate that is relative to the origin of the parent window.

Implementation Specifics

This protocol is part of AIXwindows Run Time Environment in AIXwindows Environment/6000.

Related Information

The **XCreateSimpleWindow** subroutine, **XCreateWindow** subroutine, **ConfigureWindow** protocol request.

DeleteProperty Protocol Request

Purpose

Deletes the property from the window.

Protocol Format

Window: **WINDOW**
Property: **ATOM**

Description

The **DeleteProperty** protocol request deletes a specified property from a window. Unless the specified property does not exist, this request generates a **PropertyNotify** event on the window.

Parameters

Window	Specifies a window ID.
Property	Specifies the atom corresponding to the property to be deleted.

Implementation Specifics

This protocol is part of AIXwindows Run Time Environment in AIXwindows Environment/6000.

Related Information

The **XDeleteProperty** subroutine.

DestroySubwindows Protocol Request

Purpose

Deletes all subwindows of a specified window.

Protocol Format

Window: **WINDOW**

Description

The **DestroySubwindows** protocol request generates a **DestroyWindow** protocol request, in bottom-to-top stacking order, on all children of a specified window.

Parameter

Window Specifies a window ID.

Implementation Specifics

This protocol is part of AIXwindows Run Time Environment in AIXwindows Environment/6000.

Related Information

The **XDestroySubwindows** subroutines.

DestroyWindow Protocol Request

Purpose

Deletes a window and all its inferiors.

Protocol Format

Window: **WINDOW**

Description

The **DestroyWindow** protocol request deletes a specified window and all its inferiors.

If the specified window is mapped, an **UnmapWindow** protocol request occurs automatically before the specified window and inferiors are deleted. The **DestroyNotify** events are generated for each inferior, then for the specified window. The ordering among siblings and subhierarchies is not otherwise constrained.

The **DestroyWindow** protocol request performs normal exposure processing on formerly obscured windows.

If the window specified is a root window, this request has no effect.

Field

Window	Specifies the window ID of the window to be destroyed.

Implementation Specifics

This protocol is part of AIXwindows Run Time Environment in AIXwindows Environment/6000.

Related Information

The **XDestroyWindow** subroutine.

FillPoly Protocol Request

Purpose

Fills the region defined by a set of points.

Protocol Format

Drawable: **DRAWABLE**
GraphicsContext: **GCONTEXT**
Shape: **{Complex, Nonconvex, Convex}**
CoordinateMode: **{Origin, Previous}**
Points: **LISTofPOINT**

Description

The **FillPoly** protocol request fills the region defined by the specified set of points. The set of points is closed automatically if the last set of points in the list does not coincide with the first set of points. No pixel of the region is drawn more than once.

The first point is always relative to the origin of the *Drawable* field. The remaining points are relative to the origin or to the previous point (**Previous**) depending on the *CoordinateMode* field.

The **FillPoly** protocol request uses the **GC** fields *function, plane_mask, fill_style, fill_rule, subwindow_mode, clip_x_origin, clip_y_origin,* and *clip_mask*. This protocol also uses the **GC** mode-dependent fields *foreground, background, tile, stipple, ts_x_origin,* and *ts_y_origin*.

Fields

Drawable — Specifies the drawable.

GraphicsContext — Specifies the graphics context.

Shape — Defines the path so the X Server can improve drawing performance. This field may be used by the server to improve performance. If the *Shape* field is the following:

Complex — Indicates that the path may self-intersect.

Nonconvex — Indicates that the path does not self-intersect, but the *Shape* field is not completely convex. If the **Nonconvex** value is specified for a self-intersecting path, the graphics results are undefined.

If the **Nonconvex** or **Complex** value is known by the client, specifying the **Nonconvex** value over the **Complex** value may improve performance.

Convex — Indicates that the path is wholly convex.

If the **Nonconvex** or **Complex** value is known by the client, specifying the **Convex** value may improve performance.

If the **Convex** value is specified for a path that is not convex, the graphics results are undefined.

CoordinateMode	Specifies if a point is relative to the origin of the drawable or to a previous point.
Points	Specifies the points in a polygon.

Implementation Specifics

This protocol is part of AIXwindows Run Time Environment in AIXwindows Environment/6000.

Related Information

The **XFillPolygon** subroutine.

ForceScreenSaver Protocol Request

Purpose

Activates the screen-saver function.

Protocol Format.

Mode: {**Activate, Reset**}

Description

The **ForceScreenSaver** protocol request activates the screen saver function if the *Mode* field is the **Activate** value and the screen saver function is currently deactivated, even if screen-saver has been disabled with a time out value of 0.

If the *Mode* field is **Reset** and the screen saver function is currently enabled, then the screen saver function is deactivated (if it was activated) and the activation timer is reset to its initial state as if device input had just been received.

Field

Mode	Specifies the value for the screen-saver function.

Implementation Specifics

This protocol is part of AIXwindows Run Time Environment in AIXwindows Environment/6000.

Related Information

The **XActivateScreenSaver**subroutine, **XForceScreenSaver** subroutine, **XResetScreenServer** subroutine.

FreeColormap Protocol Request

Purpose

Deletes the association between the resource ID and the colormap, and frees the colormap storage.

Protocol Format

Colormap: **COLORMAP**

Description

The **FreeColormap** protocol request deletes the association between the resource ID and the colormap and frees the colormap storage. If the colormap is an installed colormap for a screen, it is uninstalled.

If the *Colormap* field is defined by the **CreateWindow** protocol request or the **ChangeWindowAttributes** protocol request, as the colormap for a window, the colormap for the window is changed to the value of **None**, and a **ColormapNotify** event is generated.

If the *Colormap* field of a window is set to the value of **None**, the colors displayed for the window are not defined by the protocol request.

The **FreeColormap** protocol request has no effect on a default colormap for a screen.

Field

Colormap	Specifies the colormap.

Implementation Specifics

This protocol is part of AIXwindows Run Time Environment in AIXwindows Environment/6000.

Related Information

The **UninstallColormap** protocol.

The **XFreeColormap** subroutine.

FreeColors Protocol Request

Purpose

Frees all pixel parameters.

Protocol Format

Colormap: **COLORMAP**
Pixels: **LISTofCARD32**
PlaneMask: **CARD32**

Description

The **FreeColors** protocol request frees all pixels in the list of the *Pixels* field allocated by the client with the **AllocColor** protocol request, the **AllocNamedColor** protocol request, the **AllocColorCells** protocol request, and the **AllocColorPlanes** protocol request.

The *PlaneMask* field should not have any bits in common with the any of the pixels. The set of all the *Pixels* field is produced by ORing together subsets of the *PlaneMask* field with the *Pixels* field.

Freeing an individual pixel obtained from the **AllocColorPlanes** protocol request may not actually allow its reuse until all of its related *Pixels* fields are also freed.

All specified pixels allocated by the client in the *Colormap* field, 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 in the *Colormap* field. 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. If more than one pixel is in error, it is arbitrary as to which pixel is reported.

Fields

Colormap	Specifies the colormap.
Pixels	Specifies the list of pixels to be freed.
PlaneMask	Specifies the bit mask describing which planes are to be modified.

Implementation Specifics

This protocol is part of AIXwindows Run Time Environment in AIXwindows Environment/6000.

Related Information

The **XFreeColors** subroutine.

FreeCursor Protocol Request

Purpose

This protocol deletes the association between the resource ID and the cursor.

Protocol Format

Cursor: **CURSOR**

Description

The **FreeCursor** protocol request deletes the association between the resource ID and the cursor. The cursor storage is freed when no other resource references it.

Field

Cursor	Specifies the cursor to be freed.

Implementation Specifics

This protocol is part of AIXwindows Run Time Environment in AIXwindows Environment/6000.

Related Information

The **XFreeCursor** subroutine.

FreeGC Protocol Request

Purpose

Deletes the resource ID and hte *GraphicsContext* field association.

Protocol Format

GraphicsContext: **GCONTEXT**

Description

The **FreeGC** protocol request deletes the association between the resource ID and the *GraphicsContext* field and deletes the *GraphicsContext* field.

Field

GraphicsContext	Specifies the graphics context.

Error Code

GraphicsContext

Implementation Specifics

This protocol is part of AIXwindows Run Time Environment in AIXwindows Environment/6000.

Related Information

The **XFreeGC** subroutine.

FreePixmap Protocol Request

Purpose

Deletes the resource ID and the *Pixmap* field association.

Protocol Format

Pixmap: **PIXMAP**

Description

The **FreePixmap** protocol request deletes the association between the resource ID and the *Pixmap* field. The *Pixmap* field storage will be freed when no other resource references it.

Field

Pixmap	Specifies the pixmap to be freed.

Implementation Specifics

This protocol is part of AIXwindows Run Time Environment in AIXwindows Environment/6000.

Related Information

The **XFreePixmap** subroutine.

GetAtomName Protocol Request

Purpose

Returns the name for a specified atom.

Protocol Format

Atom: **ATOM**
=>
Name: **STRING8**

Description

The **GetAtomName** protocol request returns the name for a specified atom.

Parameters

Atom	Specifies an atom.
Name	Returns the name of the specified atom.

Error Code

Atom

Implementation Specifics

This protocol is part of AIXwindows Run Time Environment in AIXwindows Environment/6000.

Related Information

The **XGetAtomName** subroutine.

GetFontPath Protocol Request

Purpose

Returns the search path for fonts.

Protocol Format

=>
Path: **LISTofSTRING8**

Description

The **GetFontPath** protocol request returns the current search path for fonts.

Field

Path	Specifies the font path.

Implementation Specifics

This protocol is part of AIXwindows Run Time Environment in AIXwindows Environment/6000.

Related Information

The **XGetFontPath** subroutine.

GetGeometry Protocol Request

Purpose

Returns the root and current geometry of a specified drawable.

Protocol Format

Drawable: **DRAWABLE**
=>
Root: **WINDOW**
Depth: **CARD8**
X, *Y*: **INT16**
Width, *Height*, *BorderWidth*: **CARD16**

Description

The **GetGeometry** protocol request returns the root and current geometry of a specified drawable.

The *Depth* field returns the number of bits per pixel for the drawable.

For a pixmap:

- The *X*, *Y*, and *BorderWidth* fields return a value of 0.

For a window:

- The *X* and *Y* fields return the coordinate values of the upper left outer corner of the window relative to parent origin.
- The *Width* and *Height* fields return the inside size of the window, not including its border.

The **GetGeometry** protocol request can specify an **InputOnly** window as the specified drawable.

Fields

Drawable	Specifies a pixmap or a window ID.
Root	Returns the window ID of the drawable's root window.
Depth	Returns the depth of the specified drawable in bits per pixel.
X	Returns the x coordinate of the upper-left outer corner.
Y	Returns the y coordinate of the upper-left outer corner.
Width	Returns the width of the specified drawable excluding borders.
Height	Returns the height of the specified drawable excluding borders.
BorderWidth	Returns the border width of the specified drawable.

Error Code

Drawable

Implementation Specifics

This protocol is part of AIXwindows Run Time Environment in AIXwindows Environment/6000.

Related Information

The **XGetGeometry** subroutine, **XGetWindowAttributes** subroutine.

GetImage Protocol Request

Purpose

Returns the contents of the given rectangle of the drawable in the specified format.

Protocol Format

Drawable: **DRAWABLE**
X, Y: **INT16**
Width, Height: **CARD16**
PlaneMask: **CARD32**
Format: {**XYPixmap, ZPixmap**}
=>
Depth: **CARD8**
Visual: **VISUALID** or **None**
Data: **LISTofBYTE**

Description

The **GetImage** protocol request returns the contents of the given rectangle of the specified drawable in the specified format.

The *X* and *Y* field coordinates, which are relative to the origin of the *Drawable* field, define the upper left corner of the rectangle.

If the *Format* field is the **XYPixmap** value, only the bit planes specified in the *PlaneMask* field are transmitted with the planes appearing from most significant to least significant in bit order.

If the *Format* field is the **ZPixmap** value, the bits in all planes not specified in the *PlaneMask* field are transmitted as the values of 0.

Range checking is not performed on the *PlaneMask* field. Therefore, extraneous bits are ignored.

The *Depth* field returned is the *Depth* field specified when the *Drawable* field was created. It is the depth specified in a **FORMAT** structure in the connection setup, not a bits-per-pixel component.

If the *Drawable* field is a pixmap, the *Visual* field is the value of **None**, and the given rectangle must be contained wholly within the pixmap (or a **Match** error results).

If the *Drawable* field is a window, its *Visual* field is returned, and the window must be viewable. It must be the case that, if there were no inferiors or overlapping windows, the specified rectangle of the window would 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 protocol request.

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.

This protocol request is intended for rudimentary hardcopy support.

Fields

Drawable	Specifies the drawable.
X	Defines the upper left corner of the rectangle in conjunction with the *Y* field.
Y	Defines the upper left corner of the rectangle in conjunction with the *X* field.
Width	Specifies the width of the area to be returned.
Height	Specifies the height of the area to be returned.
PlaneMask	Specifies the planes to be returned.
Format	Specifies the format for the image. This field can be either **XYPixmap** or **ZPixmap**.
Depth	Specifies the depth of the window.
Visual	Specifies the visual ID of the window if the *Drawable* field is a window. If the *Drawable* field is a pixmap, this field is the value of **None**.
Data	Specifies the returned image.

Error Codes

Drawable

Match

Value

Implementation Specifics

This protocol is part of AIXwindows Run Time Environment in AIXwindows Environment/6000.

Related Information

The **XGetImage** subroutine.

GetInputFocus Protocol Request

Purpose

Returns the current focus state.

Protocol Format

=>
Focus: **WINDOW** or **POINTERROOT** or **NONE**
RevertTo: {**Parent, PointerRoot, None**}

Description

The **GetInputFocus** protocol request returns the current focus state.

Fields

Focus	Specifies the focus window.
RevertTo	Specifies the revert-to value should a window become not viewable.

Implementation Specifics

This protocol is part of AIXwindows Run Time Environment in AIXwindows Environment/6000.

Related Information

The **XGetInputFocus**, **XSync** subroutine.

GetKeyboardControl Protocol Request

Purpose

Returns the current control values for keyboard settings.

Protocol Format

=>
KeyClickPercent: **CARD8**
BellPercent: **CARD8**
BellPitch: **CARD16**
BellDuration: **CARD16**
LedMask: **CARD32**
GlobalAutoRepeat: **{On, Off}**
AutoRepeats: **LISTofCARD8**

Description

The **GetKeyboardControl** protocol request returns the current control values for the keyboard.

For the LEDs, the least significant bit of the *LedMask* field corresponds to LED one. Each one bit in the *LedMask* field indicates that an LED is lit.

The value of the *AutoRepeats* field is a bit vector. Each one bit indicates that *AutoRepeat* field is enabled for the corresponding key. The vector is represented as 32 bytes.

Byte n (from 0) contains the bits for the 8n to 8n+7 keys, with the least significant bit in the byte representing the 8n key.

Fields

KeyClickPercent	Specifies the key click percentage.
BellPercent	Determines the volume of the ring on the bell on the keyboard.
BellPitch	Specifies the pitch of the bell.
BellDuration	Specifies the duration of the bell.
LedMask	Indicates which LEDs are lit.
GlobalAutoRepeat	Determines whether an autorepeat is enabled for all the keys.
AutoRepeats	Indicates which keys have autorepeat enabled.

Implementation Specifics

This protocol is part of AIXwindows Run Time Environment in AIXwindows Environment/6000.

Related Information

The **XGetKeyboardControl** subroutine.

GetKeyboardMapping Protocol Request

Purpose

Returns the symbols for the specified key codes.

Protocol Format

FirstKeycode: **KEYCODE**
Count: **CARD8**

=>

KeysymsPerKeycode: **CARD8**
Keysyms: **LISTofKEYSYM**

Description

The **GetKeyboardMapping** protocol request returns the symbols for the specified number of keycodes, starting with the specified keycode.

The *FirstKeycode* field must be greater than or equal to the *MinimumKeycode* field as it was returned in the connection setup. The result of

FirstKeycode + count –1

must be less than or equal to the *MaximumKeycode* field as it was returned in the connection setup.

The number of elements in the *Keysyms* field list is

count * KeysymsPerKeycode

and the KEYSYM *Number* n (counting from 0) for the key code k has an index (counting from 0) of

```
(K - FirstKeycode) * KeysymsPerKeycode + N
```

in the *Keysyms* field list.

The *KeysymsPerKeycode* field 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 key codes.

Fields

FirstKeycode	Specifies the first key code.
Count	Sets the number of elements in the keysyms list.
KeysymsPerKeycode	Value chosen by the server to be large enough to report all requested symbols.
Keysyms	Returns the symbols for specified key codes.

Implementation Specifics

This protocol is part of AIXwindows Run Time Environment in AIXwindows Environment/6000.

Related Information

The **XGetKeyboardMapping** subroutine.

GetModifierMapping Protocol Request

Purpose

Returns the key codes for keys being used as modifiers.

Protocol Format

=>
KeycodesPerModifier: **CARD8**
Keycodes: **LISTofKEYCODE**

Description

The **GetModifierMapping** protocol request returns the key codes of the keys being used as modifiers.

The number of key codes in the list is 8* *KeycodesPerModifier* field. The key codes are divided into eight sets, with each set containing *KeycodesPerModifier* field elements. The sets are assigned in the modifiers in the following order: the **Shift, Lock, Control, Mod1, Mod2, Mod3, Mod4, and Mod5** values in order.

The *KeycodesPerModifier* field is chosen arbitrarily by the server. The value of 0s are used to fill in unused elements within each set. If only the values of 0 are given in a set, the use of the corresponding modifier has been disabled. The order of the key codes within each set is chosen arbitrarily by the server.

Fields

KeycodesPerModifier	Specifies the number of key codes per modifier.
Keycodes	Specifies the key codes being used as modifiers.

Implementation Specifics

This protocol is part of AIXwindows Run Time Environment in AIXwindows Environment/6000.

Related Information

The **XGetModifierMapping** subroutine.

GetMotionEvents Protocol Request

Purpose

Returns all events in the motion history buffer.

Protocol Format

Start, Stop: **TIMESTAMP** or **CURRENTTIME**
Window: **WINDOW**
=>
Events: **LISTofTIMECOORD**
where:
TIMECOORD: {*X, Y*: **CARD16**
Time: **TIMESTAMP**}

Description

The **GetMotionEvents** protocol request returns all events in the motion history buffer that:

- Occurred between specified start and stop times, inclusive.
- Have coordinates lying within the specified window, including borders, at its present placement.

The x and y coordinate values are reported relative to the origin of the window.

No events are returned if:

- The time specified for the *Start* field is later than the time specified for the *Stop* field.
- The time specified for the *Start* field is in the future.

If the time specified for the *Stop* field is in the future, it is equivalent to specifying a **CurrentTime** value.

Fields

Start	Specifies a start time.
Stop	Specifies a stop time.
Window	Specifies a window ID.
Events	Returns a list of events.
X	Returns the x coordinate.
Y	Returns the y coordinate.
Time	Returns a timestamp.

Implementation Specifics

This protocol is part of AIXwindows Run Time Environment in AIXwindows Environment/6000.

Related Information

The **XGetMotionEvents** subroutine.

GetPointerControl Protocol Request

Purpose

Returns the acceleration and threshold fields for the pointer.

Protocol Format

=>
AccelerationNumerator, AccelerationDenominator: **CARD16**
Threshold: **CARD16**

Description

The **GetPointerControl** protocol request returns the current acceleration and the *Threshold* field for the pointer.

Fields

AccelerationNumerator	Expresses the numerator of the acceleration multiplier.
AccelerationDenominator	Expresses the denominator of the acceleration multiplier.
Threshold	Specifies the normal acceleration of the pointer in pixels at one time.

Implementation Specifics

This protocol is part of AIXwindows Run Time Environment in AIXwindows Environment/6000.

Related Information

The **XForceScreenSaver, XGetPointerControl** subroutines.

GetPointerMapping Protocol Request

Purpose

Returns the current mapping of the pointer.

Protocol Format

Map: **LISTofCARD8**

Description

The **GetPointerMapping** protocol request returns the current mapping of the pointer. Elements of the list are indexed starting from 1. The length of the list indicates the number of physical buttons.

The nominal mapping for a pointer is the identity mapping: map[i]=i.

Field

Map	Specifies the map call for the pointer.

Implementation Specifics

This protocol is part of AIXwindows Run Time Environment in AIXwindows Environment/6000.

Related Information

The **XGetPointerMapping** subroutine.

GetProperty Protocol Request

Purpose

Returns the value of a property for a specified window.

Protocol Format

Window: **WINDOW**
Property: **ATOM**
Type: **ATOM** or **AnyPropertyType**
LongOffset, *LongLength*: **CARD32**
Delete: **BOOL**
=>
Type: **ATOM** or **None**
Format: **{0, 8, 16, 32}**
BytesAfter: **CARD32**
Value: **LISTofINT8** or **LISTofINT16** or **LISTofINT32**

Description

The **GetProperty** protocol request returns the value of a property for a specified window.

- If the specified property does not exist for the specified window:
  - The *Type* field returns a value of **None**.
  - The *Format* and *BytesAfter* fields return a value of 0.
  - The *Value* field is empty.
  - The *Delete* field is ignored.
- If the specified property exists but its type does not match the type specified:
  - The *Type* field returns the actual type of the property.
  - The *Format* field returns the nonzero value of the actual format of the property.
  - The *BytesAfter* field returns the length of the property in bytes, even if the value returned for the *Format* field is **16** or **32**.
  - The *Value* field is empty.
  - The *Delete* field is ignored
- If the specified property exists and its actual type matches the type specified or **AnyPropertyType** is specified for the *Type* field:
  - The *Type* field returns the actual type of the property.
  - The *Format* field returns the nonzero value of the actual format of the property.
  - The *BytesAfter* and *Value* fields are returned as follows:

```
N = actual length of the stored property in bytes
    (even if the Format field returns 16 or 32)
I = 4 * LongOffset
```

```
T = N - I

L = MINIMUM(T, 4 * LongLength)

A = N - (I + L)
```

Indexing from zero, the value returned by the *Value* field starts at byte index `I` in the property; `L` equals the length in bytes of this value.

If a value is specified for the *LongOffset* field such that the value of `L` is a negative, a **Value** error occurs.

BytesAfter field returns `A`, giving the number of trailing unread bytes in the stored property.

- If a value of **True** is specified for *Delete* and *BytesAfter* fields returns to the value of 0, the property is deleted from the window and a **PropertyNotify** event is generated on the window.

Fields

Window	Specifies a window ID.
Property	Specifies the atom corresponding to the property to be returned.
Type	Specifies the atom type.
LongOffset	Specifies the offset in the property where the data is to be retrieved.
LongLength	Specifies the length in 32-bit multiples of the data to be retrieved.
Delete	Specifies whether to delete the specified property.
Type	Returns the property type.
Format	Returns the property format.
BytesAfter	Returns the number of trailing unread bytes in the property.
Value	Returns the property length in bytes.

Error Codes

Atom

Value

Window

Implementation Specifics

This protocol is part of AIXwindows Run Time Environment in AIXwindows Environment/6000.

Related Information

The **XFetchBytes** subroutine, **XFetchName** subroutine, **XGetIconSizes** subroutine., **XGetNormalHints** subroutine, **XGetSizeHints** subroutine, **XGetWMHints** subroutine, **XGetWindowProperty** subroutine, **XGetZoomHints** subroutine.

GetScreenSaver Protocol Request

Purpose

Returns the current screen saver control values.

Protocol Format

=>
Timeout, Interval: **CARD16**
PreferBlanking: {**Yes**, **No**}
AllowExposures: {**Yes**, **No**}

Description

The **GetScreenSaver** protocol request returns the current screen saver control values.

Fields

Timeout	Specifies the timeout value.
Interval	Specifies the interval between screen saver invocations.
PreferBlanking	Specifies the current screen blanking preference value.
AllowExposures	Specifies the current screen save control value.

Implementation Specifics

This protocol is part of AIXwindows Run Time Environment in AIXwindows Environment/6000.

Related Information

The **XGetScreenSaver** subroutine.

GetSelectionOwner Protocol Request

Purpose

Returns the current owner window of a specified selection.

Protocol Format

Selection: **ATOM**
=>
Owner: **WINDOW** or **None**

Description

The **GetSelectionOwner** protocol request returns the current owner window of a specified selection if any. If the *Owner* field returns a value of **None**, there is no owner for the selection.

Fields

Selection	Specifies a selection.
Owner	Returns the owner of the specified selection.

Error Code

Atom

Implementation Specifics

This protocol is part of AIXwindows Run Time Environment in AIXwindows Environment/6000.

Related Information

The **XGetSelectionOwner** subroutine.

GetWindowAttributes Protocol Request

Purpose

Returns the current attributes of a window.

Protocol Format

Window: **WINDOW**
=>
Visual: **VISUALID**
Class: **{InputOutput, InputOnly}**
BitGravity: **BITGRAVITY**
WindowGravity: **WINGRAVITY**
BackingStore: **{NotUseful, WhenMapped, Always}**
BackingPlanes: **CARD32**
BackingPixel: **CARD32**
SaveUnder: **BOOL**
Colormap: **COLORMAP** or **None**
MapIsInstalled: **BOOL**
MapState: **{Unmapped, Unviewable, Viewable}**
AllEventMasks, YourEventMask: **SETofEVENT**
DoNotPropagateMask: **SETofDEVICEEVENT**
OverrideRedirect: **BOOL**

Description

The **GetWindowAttributes** protocol request returns the current attributes for a specified window.

If a value of **Unviewable** is returned for the *MapState* field, the window is mapped but an ancestor is unmapped.

The *AllEventMasks* field returns the inclusive OR of all event masks selected on the window by clients. The *YourEventMask* field returns the event mask selected by the querying client.

Fields

Window	Specifies a window for which to get current attributes.
Visual	Returns the visual ID.
Class	Returns the window class as **InputOutput, InputOnly,** or **CopyFromParent.**
BitGravity	Returns the bit gravity value as the **Forget, Static, NorthWest, North, NorthEast, West, Center, East, SouthWest, South,** or **SouthEast** values.
WindowGravity	Returns the window gravity value as the **Unmap, Static, NorthWest, North, NorthEast, West, Center, East, SouthWest, South,** or **SouthEast** values.
BackingStore	Returns the backing store value as the **WhenMapped, Always,** or **NotUseful** values.
BackingPlanes	Returns which bit planes of the window hold dynamic data.

BackingPixel	Returns what value to use in planes not covered by backing planes.
SaveUnder	Returns the value of **True** or **False**, if set to save the content of windows obscured when the specified window is mapped.
Colormap	Returns a colormap ID or the value of **None.**
MapIsInstalled	Returns installed status of a colormap.
MapState	Returns the map state as the **Unmapped, Unviewable,** or **Viewable** values.
AllEventMasks	Returns the inclusive OR of all event masks.
YourEventMasks	Returns the event mask.
DoNotPropagateMask	Returns which events are not to be propagated to ancestor windows.
OverrideRedirect	Returns whether the **SubstructureRedirect** events are to be overridden.

Error Code

Window

Implementation Specifics

This protocol is part of AIXwindows Run Time Environment in AIXwindows Environment/6000.

Related Information

The **XGetWindowAttributes** subroutine.

GrabButton Protocol Request

Purpose

Establishes a passive grab on a button-key combination.

Protocol Format

Modifiers: **SETofKEYMASK** or **AnyModifier**
Button: **BUTTON** or **AnyButton**
GrabWindow: **WINDOW**
OwnerEvents: **BOOL**
EventMask: **SETofPOINTEREVENT**
PointerMode, KeyboardMode: {**Synchronous, Asynchronous**}
ConfineTo: **WINDOW** or **None**
Cursor: **CURSOR** or **None**

Description

The **GrabButton** protocol request establishes a passive grab on a button-key combination. This request overrides all previous passive grabs by the same client on the same button/key combinations on the same window.

If all of the following are true:

- The pointer is not grabbed.
- The specified button is logically pressed when the specified modifier keys are logically down, and no other buttons or modifier keys are logically down.
- The window specified by *GrabWindow* field contains the pointer.
- The window specified by *ConfineTo* field is viewable.
- A passive grab on the same button-key combination does not exist on any ancestor of a window specified in the *GrabWindow* field.

then the following occurs:

- The pointer is actively grabbed.
- A **ButtonPress** event is reported.
- The last pointer-grab time is set to the time at which the button was pressed, as reported by the **ButtonPress** event.

The active grab is terminated when all buttons are released without regard to the logical state of modifier keys. The logical state of a device may lag the physical state if device event processing is frozen.

If the *Modifier* field is set to the **AnyModifier** type, the request is issued for all possible modifier key combinations, including the combination of no modifier keys. It is not required that all specified modifier keys have currently assigned key codes.

If the *Button* field is set to the **AnyButton** type, the request is issued for all possible buttons. It is not required that a specified button be currently assigned to a physical button.

If another client has already issued a **GrabButton** protocol request with the same button-key combination, and on the same window, an **Access** error is generated.

If there is a conflicting grab when the *Modifier* field is set to the **AnyModifier** typeor the *Button* field is set to the **AnyButton** type, no grabs are established, and an **Access** error is generated.

The **GrabButton** protocol request has no effect on an active grab.

See the **GrabPointer** protocol request for further discussion of values.

Fields

Modifiers	Specifies a modifier key combination or the **AnyModifier** type.
Button	Specifies a physical button or the **AnyButton** type.
GrabWindow	Specifies a window ID.
OwnerEvents	Specifies whether to report events to the client.
EventMask	Specifies the pointer event.
PointerMode	Specifies asynchronous or synchronous mode.
KeyboardMode	Specifies asynchronous or synchronous mode.
ConfineTo	Specifies a window ID.
Cursor	Specifies a cursor or the value of **None**.

Error Codes

Access

Cursor

Value

Window

Implementation Specifics

This protocol is part of AIXwindows Run Time Environment in AIXwindows Environment/6000.

Related Information

The **XGrabButton** subroutine.

GrabKey Protocol Request

Purpose

Establishes a passive grab on the keyboard.

Protocol Format

Key: **KEYCODE** or **AnyKey**
Modifiers: **SETofKEYMASK** or **AnyModifier**
GrabWindow: **WINDOW**
OwnerEvents: **BOOL**
PointerMode, KeyboardMode: {**Synchronous, Asynchronous**}

Description

The **GrabKey** protocol request establishes a passive grab on the keyboard. This request overrides all previous passive grabs by the same client on the same key combinations on the same window.

If all of the following are true:

- The keyboard is not grabbed.
- The specified key, which can be a modifier key, is logically pressed when the specified modifier keys are logically down, and no other modifier keys are logically down.
- The window specified by the *GrabWindow* field is the focus window, or is an ancestor of 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 any ancestor of the window specified by the *GrabWindow* field.

then the following occurs:

- The keyboard is actively grabbed.
- A **KeyPress** event is reported.
- The last keyboard-grab time is set to the time at which the key was pressed, as reported by the **KeyPress** event.

The active grab is terminated when the specified key is released, without regard to the logical state of modifier keys. The logical state of a device may lag the physical state if device event processing is frozen.

A modifier set to the **AnyModifier** value is equivalent to issuing the request for all possible modifier key combinations, including the combination of no modifier keys. It is not required that all specified modifier keys have currently assigned key codes.

A *Key* field of the **AnyKey** value is equivalent to issuing the request for all possible keys. Otherwise, the key must be in the range specified by the *min_keycode* and *max_keycode* fields at the connection setup.

If there is a conflicting grab when *Modifiers* field is set to the **AnyModifier** value or *Key* field is set to the **AnyKey** value, no grabs are established, and an **Access** error is generated.

See the **GrabKeyboard** protocol request for further discussion of values.

Fields

Key — Specifies a physical key, or the **AnyKey** value.

Modifiers	Specifies a modifier key combination, or the **AnyModifier** value.
GrabWindow	Specifies a window ID.
OwnerEvents	Specifies whether to report events to the client.
PointerMode	Specifies asynchronous or synchronous mode.
KeyboardMode	Specifies asynchronous or synchronous mode.

Error Codes

Access

Value

Window

Implementation Specifics

This protocol is part of AIXwindows Run Time Environment in AIXwindows Environment/6000.

Related Information

The **XGrabKey** subroutine.

GrabKeyboard Protocol Request

Purpose

Grabs control of the keyboard.

Protocol Format

GrabWindow: **WINDOW**
OwnerEvents: **BOOL**
PointerMode, KeyboardMode: {**Synchronous, Asynchronous**}
Time: **TIMESTAMP** or **CurrentTime**
=>
Status: {**Success, AlreadyGrabbed, Frozen, InvalidTime, NotViewable**}

Description

The **GrabKeyboard** protocol request actively grabs control of the keyboard. Further key events are reported only to the grabbing client. This protocol overrides any active keyboard grab by this client.

This protocol generates the **FocusIn** and the **FocusOut** events.

Fields

GrabWindow — Specifies a window ID.

OwnerEvents — Specifies whether to report events to the client or with respect to the *GrabWindow* field. The *OwnerEvents* field is specified as the following:

False — All generated keyboard events are reported with respect to the window specified by the *GrabWindow* field.

True — Generated keyboard events that would normally be reported to this client continue to be reported as usual. Other key events are reported with respect to the window specified by the *GrabWindow* field.

Both **KeyPress** and **KeyRelease** events are always reported, independent of any event selection made by the client.

PointerMode — Specifies asynchronous or synchronous mode. The *PointerMode* field is specified as the following:

Asynchronous — Pointer event processing continues normally.

Synchronous — For purposes of this request, the pointer appears to be frozen. No further pointer events are generated by the server until the grabbing client issues an **AllowEvents** protocol request 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.

KeyboardMode — Specifies asynchronous or synchronous mode. The *KeyboardMode* field is specified as the following:

Asynchronous — Keyboard event processing continues normally. If the keyboard is currently frozen by this client, processing of keyboard events is resumed.

Synchronous — For purposes of this request, the keyboard appears to be frozen. No further keyboard events are generated by the server until the grabbing client issues an **AllowEvents** request or until the keyboard grab is released.

Actual keyboard changes are not lost while the keyboard is frozen. They are simply queued for later processing.

Time — Specifies a timestamp or the **CurrentTime** value. If the GrabKeyboard protocol request succeeds, the last keyboard-grab time is reset to the time specified in the *Time* field. The **CurrentTime** value is the current server time; a timestamp is expressed in milliseconds.

Status — Returns the outcome of the request. The **GrabKeyboard** protocol request fails if the *Status* field returns any of the following values:

AlreadyGrabbed — The keyboard is actively grabbed by another client.**Frozen.** The keyboard is actively grabbed by another client, and is frozen.

NotViewable — The window specified in the *GrabWindow* field is not viewable.

InvalidTime — The specified time is earlier than the last keyboard-grab time or later than the current server time.

Error Codes

Value

Window

Implementation Specifics

This protocol is part of AIXwindows Run Time Environment in AIXwindows Environment/6000.

Related Information

The **XGrabKeyboard** subroutine.

GrabPointer Protocol Request

Purpose

Grabs control of the pointer.

Protocol Format

GrabWindow: **WINDOW**
OwnerEvents: **BOOL**
EventMask: **SETofPOINTEREVENT**
PointerMode, KeyboardMode: {**Synchronous, Asynchronous**}
ConfineTo: **WINDOW** or **None**
Cursor: **CURSOR** or **None**
Time: **TIMESTAMP** or **CurrentTime**
=>
Status: {**Success, AlreadyGrabbed, Frozen, InvalidTime, NotViewable**}

Description

The **GrabPointer** protocol request actively grabs control of the pointer. This request overrides any other active pointer grabs by this client. Further pointer events are reported only to this client.

The **GrabPointer** protocol request generates the **EnterNotify** and the **LeaveNotify** events.

Fields

GrabWindow — Specifies a window ID.

OwnerEvents — Specifies whether to report events to the client or the *GrabWindow* field. The *OwnerEvents* field can be specified as either of the following:

- **False** — All generated pointer events, if also selected in the *EventMask* field, are reported with respect to the window specified by the *GrabWindow* field. Unreported events are discarded.
- **True** — Generated pointer events that would normally be reported to this client continue to be reported as usual. Other generated pointer events, if also selected in the *EventMask* field, are reported with respect to the window specified by the *GrabWindow* field. Unreported events are discarded.

EventMask — Specifies the pointer event.

PointerMode — Specifies pointer mode. The *PointerMode* field can be either of the following:

- **Asynchronous** — Pointer event processing continues normally. If the pointer is currently frozen by this client, processing of pointer events is resumed.

Synchronous — For the purposes of this request, the pointer appears to be frozen. No further pointer events are generated by the server until the grabbing client issues an **AllowEvents** request 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.

KeyboardMode — Specifies the keyboard mode. The *KeyboardMode* field can be either of the following:

Asynchronous — Keyboard event processing continues normally.

Synchronous — For the purposes of this request, the keyboard appears to be frozen. No further keyboard events are generated by the server until the grabbing client issues an **AllowEvents** request or until the keyboard grab is released.

Actual keyboard changes are not lost while the pointer is frozen. They are simply queued for later processing.

ConfineTo — Specifies a window ID. If a *ConfineTo* field window is specified, the pointer position is restricted to that window. It is not necessary that the *ConfineTo* field window be related to the *GrabWindow* field. Prior to the active grab, if the pointer is not already in the window specified in the *ConfineTo* field, it is warped to the closest edge. The **EnterNotify** or the **LeaveNotify** events occur normally. If the window specified in the *ConfineTo* field is subsequently reconfigured, the pointer is warped to keep it in the window.

Cursor — Specifies a cursor or the value of **None**. If the *Cursor* field is specified, a cursor is displayed regardless of what window contains the pointer. If not specified, and the pointer is in a window specified in the *GrabWindow* field, or one of its subwindows, the normal cursor for that window is displayed. Otherwise, the cursor for the window specified by the *GrabWindow* field is displayed.

Time — If the **GrabPointer** protocol request succeeds, the last pointer-grab time is reset to the time specified in the *Time* field. The **CurrentTime** type is the current server time; a timestamp is expressed in milliseconds.

Status — Returns the outcome of the request. If the **GrabPointer** protocol request succeeds, the *Status* field returns the **Success** value. The **GrabPointer** protocol request fails if the *Status* field returns any of the following values:

AlreadyGrabbed — The pointer is actively grabbed by another client.

Frozen	The pointer is actively grabbed by another client, and is frozen.
NotViewable	A window specified in the *GrabWindow* or *ConfineTo* field is not viewable, or a window specified in the *ConfineTo* field lies completely outside the boundaries of the root window.
InvalidTime	The specified time is earlier than the last pointer-grab time or later than the current server time.

Error Codes

Cursor

Value

Window

Implementation Specifics

This protocol is part of AIXwindows Run Time Environment in AIXwindows Environment/6000.

Related Information

The **XGrabPointer** subroutine.

GrabServer Protocol Request

Purpose

Disables request processing and connection close-downs.

Description

The **GrabServer** protocol request disables processing of protocol requests and close-downs on all connections other than the one receiving this request.

Implementation Specifics

This protocol is part of AIXwindows Run Time Environment in AIXwindows Environment/6000.

Related Information

The **XGrabServer** subroutine.

ImageText16 Protocol Request

Purpose

Fills in a destination rectangle with the background pixel and paints text with the foreground pixel.

Protocol Format

Drawable: **DRAWABLE**
GraphicsContext: **GCONTEXT**
X, *Y*: **INT16**
String: **STRING16**

Description

The **ImageText16** protocol request fills in a destination rectangle with the background pixel defined in the *GraphicsContext* field 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** value as a 16-bit number that has been transmitted with the *Byte1* of the **CHAR2B** value as the most-significant byte.

The *X* and *Y* coordinates, which are relative to the origin of the *Drawable* field, specify the base line 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 *OverallWidth*, and the height is

```
font-ascent + font-descent
```

where the *OverallWidth*, *FontAscent*, and *FontDescent* fields are the same as returned by the **QueryTextExtents** protocol request using the *GraphicsContext* field and the *String* field.

The function and fill-style defined in the *GraphicsContext* field are ignored for this request; the effective function is **Copy** and the effective fill-style is **Solid**.

Fields

Drawable	Specifies the drawable.
GraphicsContext	Specifies the graphics context.
X	Specifies the x coordinate.
Y	Specifies the y coordinate.
String	Specifies the character string.

Error Codes

Drawable

GContext

Match

impiementation Specifics

This protocol is part of AIXwindows Run Time Environment in AIXwindows Environment/6000.

Related Information

The **XDrawImageString16** subroutine.

ImageText8 Protocol Request

Purpose

Fills in a destination rectangle with the background pixel and paints the text with the foreground pixel.

Protocol Format

Drawable: **DRAWABLE**
GraphicsContext: **GCONTEXT**
X, *Y*: **INT16**
String: **STRING8**

Description

The **ImageText8** protocol request fills in a destination rectangle with the background pixel defined in the *GraphicsContext* field and paints the text with the foreground pixel.

The *X* and *Y* coordinates, which are relative to the origin of the *Drawable* field, specify the base line 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 *OverallWidth*, and the height is

```
font-ascent + font-descent
```

where the *OverallWidth*, *FontAscent*, and *FontDescent* fields are the same as returned by the **QueryTextExtents** protocol request using the *GraphicsContext* field and the *String* field.

The function and fill-style defined in the *GraphicsContext* field are ignored for this protocol; the effective function is **Copy** and the effective fill-style is **Solid**.

For fonts defined with 2-byte matrix indexing, each **STRING8** byte is interpreted as a *Byte2* value of a **CHAR2B** value with a *Byte1* value of 0.

The **ImageText8** protocol request uses the following **GraphicsContext** fields, the *plane_mask, foreground, background, font, subwindow_mode, clip_x_origin, clip_y_origin,* and *clip_mask* fields.

Fields

Drawable	Specifies the drawable.
GraphicsContext	Specifies the graphics context.
X	Specifies the x coordinate.
Y	Specifies the y coordinate.
String	Specifies the character string.

Error Codes

Drawable

GContext

Match

Implementation Specifics

This protocol is part of AIXwindows Run Time Environment in AIXwindows Environment/6000.

Related Information

The **XDrawImageString** subroutine.

Enhanced X-Windows Programming Introduction
Enhanced X-Windows Protocols Overview

InstallColormap Protocol Request

Purpose

Installs colormap for the screen.

Protocol Format

Colormap: **COLORMAP**

Description

The **InstallColormap** protocol request installs this colormap for the screen. All windows associated with this colormap are displayed immediately with true colors. Depending on the server, this protocol 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 the **InstallColormap** protocol request, 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 the **UninstallColormap** protocol request 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* field is not an installed colormap, a **ColormapNotify** event is generated on every window having the *Colormap* field as an attribute. In addition, for every other colormap that is installed or uninstalled as a result of this protocol, 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.

Field

Colormap	Specifies the colormap.

Error Code

Colormap

Implementation Specifics

This protocol is part of AIXwindows Run Time Environment in AIXwindows Environment/6000.

Related Information

The **XInstallColormap** subroutine, **XQueryColor** subroutine, **XQueryColors** subroutine.

InternAtom Protocol Request

Purpose

Returns the atom for the given name.

Protocol Format

Name: **STRING8**
OnlyIfExists: **BOOL**
=>
Atom: **ATOM** or **None**

Description

The **InternAtom** protocol request returns the atom for the given name.

If the *OnlyIfExists* field is a value of **False**, the atom is created if it does not exist. The string should use the ISO Latin–1 encoding. The string is case–sensitive.

The lifetime of an atom is not tied to the interning client. Atoms remain defined until the server is reset.

Fields

Atom	Specifies the atom ID.
Name	Specifies the name associated with the atom to be returned.
OnlyIfExists	Specifies a Boolean value that indicates if the atom should be created if it does not already exist.

Error Codes

Alloc

Value

Implementation Specifics

This protocol is part of AIXwindows Run Time Environment in AIXwindows Environment/6000.

Related Information

The **XInternAtom** subroutine.

KillClient Protocol Request

Purpose

Forces a closedown of the client.

Protocol Format

Resource: **CARD32** or **AllTemporary**

Description

The **KillClient** protocol request forces a closedown of the client that created the resource for a valid *Resource* field. If the client is terminated already in the **RetainPermanent** or **RetainTemporary** mode, all client resources are deleted.

If the **AllTemporary** value is specified, the *Resource* field of all ended clients in the **RetainTemporary** value is deleted.

Field

Resource	Specifies any resource associated with the client to be deleted.

Error Code

Value

Implementation Specifics

This protocol is part of AIXwindows Run Time Environment in AIXwindows Environment/6000.

Related Information

The **XKillClient** subroutine.

ListExtensions Protocol Request

Purpose

Returns a list of all extensions supported by the X Server.

Protocol Format

=>
Names: **LISTofSTRING8**

Description

The **ListExtensions** protocol request returns a list of all extensions supported by the server.

Field

Names	Returns the list of extension names.

Implementation Specifics

This protocol is part of AIXwindows Run Time Environment in AIXwindows Environment/6000.

Related Information

The **XListExtensions** Extension.

ListFonts Protocol Request

Purpose

Returns a list of fonts that match the specified pattern.

Protocol Format

Pattern: **STRING8**
MaximumNames: **CARD16**
=>
Names: **LISTofSTRING8**

Description

The **ListFonts** protocol request returns a list of fonts (as controlled by the font search path; see the **SetFontPath** protocol request) that matches the specified pattern. The number specified in the *MaximumNames* field is the maximum number of font names returned. The *Pattern* field, which is not case-sensitive, should use the ISO Latin-1 encoding. In the *Pattern* field, the ? character (octal value 77) matches any single character, and the * character (octal value 52) matches any number of characters.

The names returned in the *Names* field are lowercase.

Fields

MaximumNames	Specifies the maximum number of names to be returned.
Pattern	Specifies the null-terminated pattern string that can contain pattern matching characters.
Names	Specifies the names of the fonts.

Implementation Specifics

This protocol is part of AIXwindows Run Time Environment in AIXwindows Environment/6000.

Related Information

The **XListFonts** subroutine.

ListFontsWithInfo Protocol Request

Purpose

Returns a list with information about each font.

Protocol Format

Pattern: **STRING8**
MaximumNames: **CARD16**
Name: **STRING8**

=>

Info: **FONTINFO**
RepliesHint: **CARD32**

where
FONTINFO: <same type definition as in the **QueryFont** protocol>

Description

The **ListFontsWithInfo** protocol request returns a list with information about each font. This list is the same as the list returned by the **QueryFont** protocol request, except that the per-character metrics are not returned.

The **ListFontsWithInfo** protocol request can generate multiple replies. With each reply, the *RepliesHint* field indicates how many more fonts are to be returned. This number is a hint only. The number of fonts returned can be larger or smaller than the number in the *RepliesHint* field. A value of 0 does not guarantee that no more fonts are to be returned. After the font replies, a reply with a 0-length name is sent to indicate the end of the reply sequence. See the **QueryFont** protocol request.

Fields

Info	Specifies font information.
MaximumNames	Specifies the maximum number of names to be returned.
Name	Specifies the name of the font whose information is in the current *Info* field.
Pattern	Specifies the null-terminated pattern string associated with the font names to be returned.
RepliesHint	Specifies how many additional fonts are to be returned.

Implementation Specifics

This protocol is part of AIXwindows Run Time Environment in AIXwindows Environment/6000.

Related Information

The **XListFontsWithInfo** subroutine.

ListHosts Protocol Request

Purpose

Returns the hosts on the access control list.

Protocol Format

=>
Mode: **{Enabled, Disabled}**
Hosts: **LISTofHOST**

Description

The **ListHosts** protocol request 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 *Hosts* field is padded to a multiple of 4 bytes.

Fields

Mode	Specifies the mode, either the **Enabled** or **Disabled** value.
Hosts	Specifies the list of hosts.

Implementation Specifics

This protocol is part of AIXwindows Run Time Environment in AIXwindows Environment/6000.

Related Information

The **XListHosts** subroutine.

ListInstalledColormaps Protocol Request

Purpose

Returns a list of the colormaps currently installed for the screen of the specified window.

Protocol Format

Window: **WINDOW**
=>
Colormaps: **LISTofCOLORMAP**

Description

The **ListInstalledColormaps** protocol request returns a list of the colormaps installed currently for the screen of the specified the *Window* field. The order of colormaps is insignificant and there is no explicit indication of the required list. See the **InstallColormap** protocol request.

Fields

Colormaps	Specifies the colormaps.
Window	Specifies the window for the colormap list.

Error Code

Window

Implementation Specifics

This protocol is part of AIXwindows Run Time Environment in AIXwindows Environment/6000.

Related Information

The **InstallColormap** protocol request.

The **XListInstalledColormaps** subroutine.

ListProperties Protocol Request

Purpose

Returns the atoms of properties currently defined on the window.

Protocol Format

Window: **WINDOW**
=>
Atoms: **LISTofATOM**

Description

The **ListProperties** protocol request returns the atoms of properties currently defined on the *Window*.

Fields

Atoms	Returns the list of atom IDs.
Window	Specifies the window.

Error Code

Window

Implementation Specifics

This protocol is part of AIXwindows Run Time Environment in AIXwindows Environment/6000.

Related Information

The **XListProperties** subroutine.

LookupColor Protocol Request

Purpose

Searches for the string name of a color.

Protocol Format

Colormap: **COLORMAP**
Name: **STRING8**
=>
ExactRed, *ExactGreen*, *ExactBlue*: **CARD16**
VisualRed, *VisualGreen*, *VisualBlue*: **CARD16**

Description

The **LookupColor** protocol searches for the string name of a color with respect to the screen associated with the *Colormap* field. It returns both the exact color values and the closest values provided by the hardware with respect to the visual type of the *Colormap* field. The *Name* field, which is not case-sensitive, should use the ISO Latin-1 encoding.

Fields

Colormap	Returns the colormap.
Name	Returns the name.
ExactRed	Returns the exact red component of the color specified in the *Name* field.
ExactGreen	Returns the exact green component of the color specified in the *Name* field.
ExactBlue	Returns the exact blue component of the color specified in the *Name* filed.
VisualRed	Returns the closest red component provided by the hardware.
VisualGreen	Returns the closest green component provided by the hardware.
VisualBlue	Returns the closest blue component provided by the hardware.

Error Codes

Colormap

Name

Implementation Specifics

This protocol is part of AIXwindows Run Time Environment in AIXwindows Environment/6000.

Related Information

The **XLookupColor** subroutine, **XParseColor** subroutine.

MapSubwindows Protocol Request

Purpose

Performs a **MapWindow** protocol request of unmapped children of the window.

Protocol Format

WindowID: **WINDOW**

Description

The **MapSubwindows** protocol request performs a **MapWindow** protocol request on all unmapped children of the *WindowID* in a top-to-bottom stacking order.

Field

WindowID Specifies the window.

Error Code

Window

Implementation Specifics

This protocol is part of AIXwindows Run Time Environment in AIXwindows Environment/6000.

Related Information

The **XMapSubwindows** subroutine.

Enhanced X-Windows Programming Introduction
Enhanced X-Windows Protocols Overview

MapWindow Protocol Request

Purpose

Maps an unmapped window.

Protocol Format

Window: **WINDOW**

Description

The **MapWindow** protocol request maps an unmapped window. If the override-redirect attribute of the *Window* is a value of **False** and another client has selected the **SubstructureRedirect** value on the parent window, 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 0 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 protocol request has no effect if the window is already mapped.

Field

Window	Specifies the window.

Error Code

Window

Implementation Specifics

This protocol is part of AIXwindows Run Time Environment in AIXwindows Environment/6000.

Related Information

The **XMapRaised** subroutine, **XMapWindow** subroutine.

Enhanced X-Windows Introduction
Enhanced X-Windows Protocols Overview

NoOperation Protocol Request

Purpose

Forces request to begin on 64-bit boundaries.

Description

This protocol request has no fields and no results, but the protocol request length field can be nonzero, allowing the protocol request to be any multiple of 4 bytes in length. The bytes contained in the protocol request are uninterpreted by the server.

The **NoOperation** protocol request can be used in its minimum 4-byte form as padding where necessary by client libraries that find it convenient to force protocol requests to begin on 64-bit boundaries.

Implementation Specifics

This protocol is part of AIXwindows Run Time Environment in AIXwindows Environment/6000.

Related Information

The **XNoOp** subroutine.

Enhanced X-Windows Introduction
Enhanced X-Windows Protocols Overview

OpenFont Protocol Request

Purpose

Loads the specified font and associates an identifier with it.

Protocol Format

FontID: **FONT**
Name: **STRING8**

Description

The **OpenFont** protocol request loads the specified font, if necessary, and associates the *FontID* field identifier with it. The font name should use the ISO Latin-1 encoding and is not case-sensitive.

Fonts are not associated with a particular screen and can be stored as a component of any graphics context.

Fields

FontID	Identifies the font.
Name	Name of the font.

Error Codes

Alloc

IDChoice

Name

Implementation Specifics

This protocol is part of AIXwindows Run Time Environment in AIXwindows Environment/6000.

Related Information

The **XLoadFont** subroutine, **XLoadQueryFont** subroutine.

Enhanced X-Windows Programming Introduction
Enhanced X-Windows Protocols Overview

PolyArc Protocol Request

Purpose

Draws circular or elliptical arcs.

Protocol Format

Drawable: **DRAWABLE**
GraphicsContext: **GCONTEXT**
Arcs: **LISTofARC**

Description

The **PolyArc** protocol request draws circular or elliptical arcs. Each *Arc* field 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* field is specified by the *Angle1* relative to the 3 o'clock position from the center of the rectangle. The path and extent of the *Arc* field is specified by the *Angle2* relative to the start of the *Arc* field. If the magnitude of the *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* field.

For an *Arc* field 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 two infinitely thin paths consisting of all points whose perpendicular distance from the path of the circle/ellipse is equal to lw/2 (which may be a fractional value) describing the *Arc* field. The *CapStyle* and *JoinStyle* are applied the same as for a line corresponding to the tangent of the circle/ellipse at the endpoint.

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:

```
SkewedAngle = Atan(tan(NormalAngle) * w/h) + Adjust
```

where SkewedAngle and NormalAngle 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 the Adjust value is as follows:

0	for NormalAngle in the range [0,PI/2]
PI	for NormalAngle in the range [PI/2,(3*PI)/2]
2*PI	for NormalAngle 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 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 0, 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 0, a horizontal or vertical line can be drawn.

Angles are computed based solely on the coordinate system, ignoring the aspect ratio.

The **PolyArc** protocol request uses the **GraphicsContext** fields *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 **GraphicsContext** mode-dependent fields *foreground, background, tile, stipple, ts_x_origin, ts_y_origin, dash_offset,* and *dashes.*

Fields

Drawable	Specifies the drawable.
GraphicsContext	Specifies the graphics context.
Arcs	Specifies the list of arcs.

Error Codes

Drawable

GContext

Match

Implementation Specifics

This protocol is part of AIXwindows Run Time Environment in AIXwindows Environment/6000.

Related Information

The **XDrawArc** subroutine, **XDrawArcs** subroutine.

Enhanced X-Windows Programming Introduction
Enhanced X-Windows Protocols Overview

PolyFillArc Protocol Request

Purpose

Fills the regions closed by the path described in the arc and arc mode.

Protocol Format

Drawable: **DRAWABLE**
GraphicsContext: **GCONTEXT**
Arcs: **LISTofARC**

Description

For each arc, the **PolyFillArc** protocol request 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 the **Chord** mode, the single line segment joining the endpoints of the arc is used. For the **PieSlice** mode, the two line segments joining the endpoints of the arc with the center point are used. The *Arcs* field are as specified in the **PolyArc** protocol request.

The arcs specified in the *Arcs* field 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.

The **PolyFillArc** protocol request uses the **GraphicsContext** fields *function, plane_mask, fill_style, arc_mode, subwindow_mode, clip_x_origin, clip_y_origin,* and *clip_mask*. It also uses the **GraphicsContext** mode-dependent fields *foreground, background, tile, stipple, ts_x_origin,* and *ts_y_origin*.

Fields

Drawable	Specifies the drawable.
GraphicsContext	Specifies the graphics context.
Arcs	Specifies the arcs to be filled.

Error Codes

Drawable

GContext

Match

Implementation Specifics

This protocol is part of AIXwindows Run Time Environment in AIXwindows Environment/6000.

Related Information

The **XFillArc** subroutine, **XFillArcs** subroutine.

Enhanced X-Windows Programming Introduction
Enhanced X-Windows Protocols Overview

PolyFillRectangle Protocol Request

Purpose

Fills the specified rectangles.

Protocol Format

Drawable: **DRAWABLE**
GraphicsContext: **GCONTEXT**
Rectangles: **LISTofRECTANGLE**

Description

The **PolyFillRectangle** protocol request fills the specified rectangles as if a four-point **FillPoly** protocol request was specified for each rectangle as follows:

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

The x and y coordinates of each *Rectangle* field, which are relative to the origin of the *Drawable* field, 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.

The **PolyFillRectangle** protocol request uses the **GraphicsContext** fields *function, plane_mask, fill_style, subwindow_mode, clip_x_origin, clip_y_origin,* and *clip_mask.* It also uses the **GraphicsContext** mode-dependent fields *foreground, background, tile, stipple, ts_x_origin,* and *ts_y_origin.*

Fields

Drawable	Specifies the drawable.
GraphicsContext	Specifies the graphics context.
Rectangles	Specifies the list of rectangles.

Error Codes

Drawable

GContext

Match

Implementation Specifics

This protocol is part of AIXwindows Run Time Environment in AIXwindows Environment/6000.

Related Information

The **XFillRectangle** soubroutines, **XFillRectangles** subroutine.

Enhanced X-Windows Programming Introduction
Enhanced X-Windows Protocols Overview

PolyPoint Protocol Request

Purpose

Combines the foreground pixel with the pixel at each point in the drawable.

Protocol Format

Drawable: **DRAWABLE**
GraphicsContext: **GCONTEXT**
CoordinateMode: **{Origin, Previous}**
Points: **LISTofPOINT**

Description

The **PolyPoint** protocol request combines the foreground pixel in the *GraphicsContext* field with the pixel at each point in the *Drawable* field. The points are drawn in the order listed in the *Points* field.

The first point is always relative to the origin of the drawable. The other points are relative either to the origin of the drawable (the **Origin** value) or to the origin of the previous point (the **Previous** value) depending on the *CoordinateMode* field.

The **PolyPoint** protocol request uses the **GraphicsContext** fields *function, plane_mask, foreground, subwindow_mode, clip_x_origin, clip_y_origin,* and *clip_mask.*

Fields

Drawable	Specifies the drawable.
GraphicsContext	Specifies the graphics context.
CoordinateMode	Specifies whether all points after the first are relative to the origin of the drawable or relative to the previous point.
Points	Specifies the points to be drawn.

Error Codes

Drawable

GContext

Value

Match

Implementation Specifics

This protocol is part of AIXwindows Run Time Environment in AIXwindows Environment/6000.

Related Information

The **XDrawPoint** subroutine, **XDrawPoints** subroutine.

Enhanced X-Windows Programming Introduction
Enhanced X-Windows Protocols Overview

PolyLine Protocol Request

Purpose

Draws lines between each pair of *Points* field.

Protocol Format

Drawable: **DRAWABLE**
GraphicsContext: **GCONTEXT**
CoordinateMode: **{Origin, Previous}**
Points: **LISTofPOINT**

Description

The **PolyLine** protocol request draws lines between each pair of points specified on the *Points* field (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 (0 line width) 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** protocol request 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 (the **Origin** value) or to the origin of the previous point (the **Previous** value) depending on the *CoordinateMode* field.

The **PolyLine** protocol request uses the **GraphicsContext** fields *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 **GraphicsContext** mode-dependent fields *foreground, background, tile, stipple, ts_x_origin, ts_y_origin, dash_offset,* and *dashes*.

Fields

Drawable	Specifies the drawable.
GraphicsContext	Specifies the graphics context.
CoordinateMode	Specifies the relation of a point either to a previous point or to the drawable's origin.
Points	Specifies the points to be drawn.

Error Codes

Drawable

GContext

Match

Value

Implementation Specifics

This protocol is part of AIXwindows Run Time Environment in AIXwindows Environment/6000.

Related Information

The **XDrawLines** subroutine, **XDrawRectangle** subroutine, **XDrawRectangles** subroutine.

Enhanced X-Windows Programming Introduction
Enhanced X-Windows Protocols Overview

PolyRectangle Protocol Request

Purpose

Draws the outlines of the specified rectangles.

Protocol Format

Drawable: **DRAWABLE**
GraphicsContext: **GCONTEXT**
Rectangles: **LISTofRECTANGLE**

Description

The **PolyRectangle** protocol request draws the outlines of the rectangles specified in the *Rectangles* field, as if a five-point **PolyLine** protocol request was specified for each rectangle as follows:

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

The x and y coordinates of each rectangle, which are relative to the origin of the drawable specified in the *Drawable* field, define the upper left corner of the *Rectangle* field.

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 several times.

The **PolyRectangle** protocol request uses the **GraphicsContext** fields *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 **GraphicsContext** mode-dependent fields *foreground, background, tile, stipple, ts_x_origin, ts_y_origin, dash_offset,* and *dashes*.

Fields

Drawable	Specifies the drawable.
GraphicsContext	Specifies the graphics context.
Rectangles	Specifies the list of rectangles.

Error Codes

Drawable

GContext

Match

Implementation Specifics

This protocol is part of AIXwindows Run Time Environment in AIXwindows Environment/6000.

Related Information

The **XDrawRectangle** subroutine, **XDrawRectangles** subroutine.

Enhanced X-Windows Programming Introduction
Enhanced X-Windows Protocols Overview

PolySegment Protocol Request

Purpose

Draws a line for each segment.

Protocol Format

Drawable: **DRAWABLE**
GraphicsContext: **GCONTEXT**
Segments: **LISTofSEGMENT**
where
SEGMENT: [x1, y1, x2, y2: **INT16**]

Description

The **PolySegment** protocol request 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.

The **PolySegment** protocol request uses the **GraphicsContext** fields *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 **GraphicsContext** mode-dependent fields *foreground, background, tile, stipple, ts_x_origin, ts_y_origin, dash_offset,* and *dashes*.

Fields

Drawable	Specifies the drawable.
GraphicsContext	Specifies the graphics context.
Segments	Specifies the line segments to be drawn.

Error Codes

Drawable

GContext

Match

Implementation Specifics

This protocol is part of AIXwindows Run Time Environment in AIXwindows Environment/6000.

Related Information

The **XDrawLine** subroutine, **XDrawSegments** subroutine.

Enhanced X-Windows Introduction
Enhanced X-Windows Protocols Overview

PolyText16 Protocol Request

Purpose

Draws text.

Protocol Format

Drawable: **DRAWABLE**
GraphicsContext: **GCONTEXT**
X, *Y*: **INT16**
Items: **LISTofTEXTITEM16**

where
TEXTITEM16: **TEXTELT16** or **FONT**
TEXTELT16: [*Delta*: **INT8**
String: **STRING16**]

Description

The **PolyText16** protocol request is the same as the **PolyText8** protocol request 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** value as a 16-bit number transmitted with the most significant byte first (*Byte1*).

Fields

Drawable	Specifies the drawable.
GraphicsContext	Specifies the graphics context.
X	Specifies the x coordinate.
Y	Specifies the y coordinate.
Items	Specifies the text items font or character string.
Delta	Specifies an additional change in the position along the x axis before the string is drawn. It is always added to the character origin.

Error Codes

Drawable

Font

GContext

Match

Implementation Specifics

This protocol is part of AIXwindows Run Time Environment in AIXwindows Environment/6000.

Related Information

The **XDrawImageString16**subroutine, **XDrawString16** subroutine, **XDrawText16** subroutine.

The **PolyText8** protocol request.

Enhanced X-Windows Introduction
Enhanced X-Windows Protocols Overview

PolyText8 Protocol Request

Purpose

Draws text.

Protocol Format

Drawable: **DRAWABLE**
GraphicsContext: **GCONTEXT**
X, *Y*: **INT16**
Items: **LISTofTEXTITEM8**

where
TEXTITEM8: TEXTELT8 or **FONT**
EXTELT8: [*Delta*: **INT8** *String*: **STRING8**]

Description

The **PolyText8** protocol request draws text. Each text item is processed in turn. A font item causes the font to be stored in the *GraphicsContext*, 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* field, specify the baseline starting position (the initial character origin).

A text element *Delta* field specifies an additional change in the position along the x axis before the string is drawn; the *Delta* field is always added to the character origin. Each character image, as defined by the font in the *GraphicsContext*, is treated as an additional mask for a fill operation on the *Drawable* field.

All contained **FONT** values 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 0.

The **PolyText8** protocol request uses the **GraphicsContext** fields *function, plane_mask, fill_style, font, subwindow_mode, clip_x_origin, clip_y_origin,* and *clip_mask*. It also uses the **GraphicsContext** mode-dependent fields *foreground, background, tile, stipple. ts_x_origin,* and *ts_y_origin*.

Fields

Drawable Specifies the drawable.

GraphicsContext Specifies the graphics context.

X Specifies the x coordinate.

Y Specifies the y coordinate.

Items Specifies the text items or character string:

- *Delta* Specifies an additional change in the position along the x axis before the string is drawn. It is always added to the character origin.
- *String* Specifies the string to be drawn.

Error Codes

Drawable

Font

GContext

Match

Implementation Specifics

This protocol is part of AIXwindows Run Time Environment in AIXwindows Environment/6000.

Enhanced X-Windows Introduction
Enhanced X-Windows Protocols Overview

PutImage Protocol Request

Purpose

Combines an image with a rectangle of the specified drawable.

Protocol Format

Drawable: **DRAWABLE**
GraphicsContext: **GCONTEXT**
Depth: **CARD8**
Width, *Height*: **CARD16**
DestinationX, *DestinationY*: **INT16**
LeftPad: **CARD8**
Format: {**Bitmap, XYPixmap, ZPixmap**}
Data: **LISTofBYTE**

Description

The **PutImage** protocol request combines an image with a rectangle of the specified *Drawable* field. The *DestinationX* and *DestinationY* field coordinates are relative to origin of the *Drawable* field.

If the value of the *Format* field is **Bitmap**, the depth must be a value of 1 (or a **Match** error results) and the image must be in **XYFormat**.

The foreground pixel in the *GraphicsContext* defines the source for one bits in the image, and the background pixel defines the source for the 0 bits.

For **XYPixmap** and **ZPixmap** formats, the depth must match the depth of the *Drawable* field (or a **Match** error results).

- For the **XYPixmap** format, the image must be in **XYFormat**.
- For the **ZPixmap** format, the image must be in **ZFormat** for the specified depth.

The first *LeftPad* field bits in each scanline are ignored by the server; the actual image begins that number of bits into the data.

- For the **ZPixmap** format, the *LeftPad* field must be a value of 0.
- For the **Bitmap** and **XYPixmap** value, the *LeftPad* field must be less than the *BitmapScanlinePad* field as specified in the server connection setup (or a **Match** error results).

The *Width* field defines the width of the actual image and does not include the *LeftPad* field.

The **PutImage** protocol request uses the **GraphicsContext** *function, plane_mask, subwindow_mode, clip_x_origin, clip_y_origin,* and *clip_mask*. It also uses the **GraphicsContext** mode-dependent *foreground* and *background*.

Fields

Drawable	Specifies the destination drawable.
GraphicsContext	Specifies the graphics context.
Depth	Specifies the depth of the actual image.

Width	Specifies the width of the actual image and does not include left-pad.
Height	Specifies the height of the actual image.
DestinationX	Specifies the *x* coordinate relative to the origin of the drawable.
DestinationY	Specifies the y coordinates relative to the origin of the drawable.
LeftPad	Specifies the left-pad bits in a scanline.
Format	Specifies the organization of pixel values for a pixmap.
Data	Specifies the data for a pixmap.

Error Codes

Drawable

GContext

Match

Value

Implementation Specifics

This protocol is part of AIXwindows Run Time Environment in AIXwindows Environment/6000.

Related Information

The **XPutImage** subroutine.

Enhanced X-Windows Introduction
Enhanced X-Windows Protocols Overview

QueryBestSize Protocol Request

Purpose

Returns the size closest to the specified size.

Protocol Format

Class: {**Cursor, Tile, Stipple**}
Drawable: **DRAWABLE**
Width, *Height*: **CARD16**
=>
Width, *Height*: **CARD16**

Description

The **QueryBestSize** protocol request returns the size closest to the specified size.

If the *Class* field is **Cursor**, this is the largest cursor that can be displayed fully on the *Drawable* (screen) field.

If the *Class* field is **Tile**, this is the size that can be tiled fastest on the *Drawable* field, which can be the screen or the window class and depth. An **InputOnly** window cannot be used as the *Drawable* field for **Tile**.

If the *Class* field is **Stipple**, this is the size that can be stippled fastest on the specified *Drawable* field, which can be the screen or the window class and depth. An **InputOnly** window cannot be used as the *Drawable* field for **Stipple**.

Fields

Class	Specifies the type of information being queried.
Drawable	Specifies the drawable.
Width	Specifies the window width.
Height	Specifies the window height.
Width	Returns the best width.
Height	Returns the best height.

Error Codes

Drawable

Match

Value

Implementation Specifics

This protocol is part of AIXwindows Run Time Environment in AIXwindows Environment/6000.

Related Information

The **XQueryBestCursor**subroutine, **XQueryBestSize**subroutine, **XQueryBestStipple** subroutine, **XQueryBestTile** subroutine.

Enhanced X-Windows Introduction
Enhanced X-Windows Protocols Overview

QueryColors Protocol Request

Purpose

Returns the color values for the specified pixels.

Protocol Format

Colormap: **COLORMAP**
Pixels: **LISTofCARD32**
=>
Colors: **LISTofRGB** where
RGB: [**red, green, blue**: **CARD16**]

Description

The **QueryColors** protocol request returns the color values stored in the *Colormap* field for the specified pixels. Values returned for an unallocated entry are not defined. A **Value** error is generated if a pixel is not a valid index into the *Colormap* field. If more than one pixel is in error, it is arbitrary as to which pixel is reported.

Fields

Colormap	Specifies the colormap.
Pixels	Specifies pixels for query.
Colors	Returns the colors stored in the colormap that correspond to the specified pixels.

Error Codes

Colormap

Value

Implementation Specifics

This protocol is part of AIXwindows Run Time Environment in AIXwindows Environment/6000.

Related Information

The **XQueryColor** subroutine, **XQueryColors** subroutine.

Enhanced X-Windows Introduction
Enhanced X-Windows Protocols Overview

QueryExtension Protocol Request

Purpose

Determines if the extension named is present.

Protocol Format

Name: **STRING8**
=>
Present: **BOOL**
MajorOpcode: **CARD8**
FirstEvent: **CARD8**
FirstError: **CARD8**

Description

The **QueryExtension** protocol request determines if the named extension is present. If the extension is present, the *MajorOpcode* field is returned, if it has one. If the extension is not available, the *MajorOpcode* field returns a value of 0. Any minor-opcode and the protocol request formats are specific to the extension. If the extension involves additional event types, the base event type code is returned. Otherwise, a value of 0 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 value of 0 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.

Fields

Name	Specifies the name of the extension being queried.
Present	Returns the boolean specifying the presence of the extension.
MajorOpcode	Returns the major opcode of the extension.
FirstEvent	Returns the base event type.
FirstError	Returns the base error type.

Implementation Specifics

This protocol is part of AIXwindows Run Time Environment in AIXwindows Environment/6000.

Related Information

The **XQueryExtension** extension subroutine.

Enhanced X-Windows Introduction
Enhanced X-Windows Protocols Overview

QueryFont Protocol Request

Purpose

Returns logical information about a font.

Protocol Format

Font: **FONTABLE**
=>
Fontinfo: **FONTINFO**
CharacterInformations: **LISTofCHARINFO**
where:

FONTINFO: [*DrawDirection*: {**LeftToRight, RightToLeft**}
MinimumCharacterOrByte2, MaximumCharacterOrByte2: **CARD16**
MinimumByte1, MaximumByte1: **CARD8**
AllCharactersExist: **BOOL**
DefaultCharacter: **CARD16**
MinimumBounds: **CHARINFO**
MaximumBounds: **CHARINFO**
FontAscent: **INT16**
FontDescent: **INT16**
Properties: **LISTofFONTPROP**]

FONTPROP: [*Name*: **ATOM**
Value: <32-*BitValue*>]

CHARINFO: [*LeftSideBearing*: **INT16**
RightSideBearing: **INT16**
CharacterWidth: **INT16**
Ascent: **INT16**
Descent: **INT16**
Attributes: **CARD16**]

Description

The **QueryFont** protocol request returns logical information about a font. If **GraphicsContext** is given, the currently contained font is used.

The *DrawDirection* field is just a hint, indicating whether most *CharacterInformations* field have a positive (the **LeftToRight** value) or a negative (the **RightToLeft** value) *CharacterWidth* field. The core protocol request defines no support for vertical text.

If the *MinimumByte1* and *MaximumByte1* fields are both the value of 0, then the *MinimumCharacterOrByte2* field specifies the linear character index corresponding to the first element of the *CharacterInformations* field, and *MaximumCharacterOrByte2* field specifies the linear character index of the last element. If either the *MinimumByte1* or the *MaximumByte1* fields are nonzero, then both the *MinimumCharacterOrByte2* and *MaximumCharacterOrByte2* fields will be less than 256, and the 2-byte character index values corresponding to the *CharacterInformations* field element `N` (counting from 0) are as follows:

```
byte1 = N/D + MinimumByte1
byte2 = N/D + MinimumCharacterOrByte2
where
D = MaximumCharacterOrByte2 - MinimumCharacterOrByte2 + 1
/ = integer division
\ = integer modulus
```

If the *CharacterInformations* field has a length of 0, then the *MinimumBounds* and *MaximumBounds* fields will be identical, and the effective *CharacterInformations* field is one filled with this *CharacterInformation* field, of the following length:

```
L = D * (MaximumByte1 – MinimumByte1 + 1)
```

All glyphs in the specified linear or matrix range have the same information as that given by the *MinimumBounds* field (and *MaximumBounds* field). If the *AllCharactersExist* field is the value of **True**, all characters in the *CharacterInformations* field have nonzero bounding boxes.

The *DefaultCharacter* field specifies the character that will be used when an undefined or non-existent character is used. Note that the *DefaultCharacter* field is a **CARD16** type (not a **CHAR2B** type); for a font using 2-byte matrix format, the *DefaultCharacter* field has the *byte1* field in the most significant byte, and the *byte2* field is in the least significant byte. If the *DefaultCharacter* field itself specifies an undefined or non-existent character, then printing is not performed for an undefined or non-existent character.

The *MinimumBounds* and *MaximumBounds* fields contain the minimum and maximum values of each individual **CHARINFO** component over all the *CharacterInformations* fields (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 the following:

```
[x + MinimumBounds.left-side-bearing, y – MaximumBounds.ascent]
```

with a width of:

```
MaximumBounds.right-side-bearing – MinimumBounds.left-side-bearing
```

and a height of:

```
MaximumBounds.ascent + MaximumBounds.descent
```

The *FontAscent* field is the logical extent of the font above the base line for determining line spacing. Specific characters can extend beyond this. The *FontDescent* field is the logical extent of the font at or below the base line, for determining line spacing. Specific characters may extend beyond this. If the base line is at the Y-coordinate `y`, then the logical extent of the font is inclusive between the Y-coordinate values (`Y – FontAscent`) and (`Y + FontDescent – 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	**INT32**	Offsets 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**	Offsets from the character origin where subscripts should begin (in pixels). If the origin is at [*X, Y*], then subscripts should begin at [*X* + SUBSCRIPT_X, *Y* + SUBSCRIPT_Y].
UNDERLINE_POSITION	**INT32**	Y offset from the baseline to the top of an underline (in pixels). If the baseline is the 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 the 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 3 o'clock position from the character origin, with positive indicating counterclockwise motion as in the **DrawArc** protocol request.
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 the Y-coordinate *Y* variable, 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** type components, then it is a rectangle with its upper-left corner at:

```
[x + LeftSideBearing, y – Ascent]
```

and a width of:

```
RightSideBearing – LeftSideBearing
```

and a height of:

```
Ascent + Descent
```

and the origin for the next character is defined to be:

```
[x + CharacterWidth, y]
```

Note that the baseline is logically viewed as being just below non-descending characters (when the *Descent* field is the value of 0, 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 the *LeftSideBearing* field is the value of 0, 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 the value of 0.

The interpretation of the per-character attributes field is server-dependent.

Fields

Font — Specifies the font.

Fontinfo — Defines the characteristics of the specified font.

CharacterInformations — If present, specifies the dimensions and attributes of font elements.

DrawDirection — Indicates whether most of the *CharacterInformations* field have a positive or negative *CharacterWidth* field metric. This hint can have the following values:

- **LeftToRight** — Indicates that most of the *CharacterInformations* field have a positive *CharacterWidth* field metric.
- **RightToLeft** — Indicates that most of the *CharacterInformations* field have a negative *CharacterWidth* field metric.

MinimumCharacterOrByte2 — Specifies the linear character index corresponding to the first element in the *CharacterInformations* field (first character).

MaximumCharacterOrByte2	Specifies the linear character index of the last element (last character).
MinimumByte1	Determines whether the *MinimumCharacterOrByte2* field and the *MaximumCharacterOrByte2* field will be less than 256 (first row that exists).
MaximumByte1	Determines whether the *MinimumCharacterOrByte2* field and the *MaximumCharacterOrByte2* field will be less than 256 (last row that exists).
AllCharactersExist	Specifies whether all characters in the *CharacterInformations* field have nonzero bounding boxes.
DefaultCharacter	Specifies the character that will be used when an undefined or non-existent character is used.
MinimumBounds	Contains the minimum value of each **CHARINFO** component over all *CharacterInformations*.
MaximumBounds	Contains the maximum value of each **CHARINFO** component over all *CharaterInformations*.
FontAscent	Returns the logical extent of the ascent metrics.
FontDescent	Returns the logical extent of the descent metrics.
Properties	Specifies the properties of the referenced font.
Name	Specifies the name of the property.
Value	Specifies the value of the property.
LeftSideBearing	Origin to left edge of raster.
RightSideBearing	Origin to right edge of raster.
CharacterWidth	Specifies the width of the specified character.
Ascent	Logical extent above baseline for spacing.
Descent	Logical extent below baseline for spacing.
Attributes	Specifies the attributes of the specified character.

Error Codes

Font

Implementation Specifics

This protocol is part of AIXwindows Run Time Environment in AIXwindows Environment/6000.

Related Information

The **XLoadQueryFont** subroutine.

The **QueryTextExtents** protocol.

QueryKeymap Protocol Request

Purpose

Returns a bit vector for the keyboard.

Protocol Format

=>
Keys: **LISTofCARD8**

Description

The **QueryKeymap** protocol request returns a bit vector for the logical state of 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 `8N` to `8N+7`, with the least significant bit in the byte representing the `8N` key.

Note: The logical state of a device (as seen by means of the protocol request) may lag behind the physical state if the device event processing is frozen.

Field

Keys Specifies an array of bytes that represents the keys.

Implementation Specifics

This protocol is part of AIXwindows Run Time Environment in AIXwindows Environment/6000.

Related Information

The **XQueryKeymap** subroutine.

QueryPointer Protocol Request

Purpose

Returns the root window and the coordinates for the current pointer position.

Protocol Format

Window: **WINDOW**
=>
Root: **WINDOW**
Child: **WINDOW** or **None**
SameScreen: **BOOL**
RootX, *RootY*, *WindowX*, *WindowY*: **INT16**
Mask: **SETofKEYBUTMASK**

Description

The **QueryPointer** protocol request returns the root window the pointer is logically on and the coordinates of the pointer relative to the origin of the root. The current logical state of the modifier keys and the buttons are also returned.

If the *SameScreen* field returns a value of **False**, the pointer is not on the same screen as specified by the *Window* field, the *Child* field has a value of **None**, and the *WindowX* and *WindowY* fields have a value of 0.

If the *SameScreen* field returns a value of **True**, the *WindowX* and *WindowY* field are the pointer coordinates relative to the origin of the specified window. The *Child* field returns the window containing the pointer, if it is contained in a child window of the specified window.

The *Mask* field returns the current logical state of the modifier keys and buttons.

Note: The logical state of a device might lag behind the physical state if device event processing is frozen.

Fields

Window	Specifies the window ID.
Root	Returns the root window the pointer is logically on.
Child	Returns the child window containing the pointer.
SameScreen	Returns the value of **True** or **False** if the pointer is on the same window as *Window* field.
RootX	Returns the x coordinate for the pointer relative to the root window.
RootY	Returns the y coordinate for the pointer relative to the root window.
WindowX	Returns the x coordinate for the pointer relative to the *Window* field.
WindowY	Returns the y coordinate for the pointer relative to the *Window* field.
Mask	Returns the current logical state of the modifier keys and buttons.

Error Code

Window

Implementation Specifics

This protocol is part of AIXwindows Run Time Environment in AIXwindows Environment/6000.

Related Information

The **XQueryPointer** subroutine.

QueryTextExtents Protocol Request

Purpose

Returns the logical extents of a specified string of characters in a specified font.

Protocol Format

Font: **FONTABLE**
String: **STRING16**
=>
DrawDirection: {**LeftToRight, RightToLeft**}
FontAscent: **INT16**
FontDescent: **INT16**
OverallAscent: **INT16**
OverallDescent: **INT16**
OverallWidth: **INT32**
OverallLeft: **INT32**
OverallRight: **INT32**

Description

The **QueryTextExtents** protocol returns the logical extents of a specified string of characters in a specified font.

If a graphics context is given, the font currently contained is used.

Values are specified for the *DrawDirection*, *FontAscent*, and *FontDescent* fields the same as for the **QueryFont** protocol.

The *OverallAscent* field is the maximum of the ascent metrics of all characters in the string; the *OverallDescent* field is the maximum of the descent metrics of all characters in the string.

The *OverallWidth* field is the sum of the character-width metrics of all characters in the string.

For each character specified in the *String* field:

- 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`.
- Let `r` be the right-side-bearing metric of the character plus `w`.

Then, the value returned for:

- The *OverallLeft* field is the minimum `l` value of all characters in the string.
- The *OverallRight* field is the maximum `r` value of all characters in the string.

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 *Byte1* as the most significant byte.

Characters with all zero metrics are ignored. If the font has no defined default character, then undefined characters in the string are also ignored.

Fields

Font	Specifies the font or the **Graphics Context** containing the font.
String	Specifies the character string.
DrawDirection	Returns the draw directions.
FontAscent	Returns the logical extent of the ascent metrics.
FontDescent	Returns the logical extent of the descent metrics.
OverallAscent	Returns the maximum of the ascent metrics.
OverallDescent	Returns the maximum of the descent metrics.
OverallWidth	Returns the overall width of the character-width metrics.
OverallLeft	Returns the left-side-bearing metrics.
OverallRight	Returns the right-side-bearing metrics.

Error Code

Font

Implementation Specifics

This protocol is part of AIXwindows Run Time Environment in AIXwindows Environment/6000.

Related Information

The **XQueryTextExtents** subroutine, **XQueryTextExtents16** subroutine.

The **QueryFont** protocol.

QueryTree Protocol Request

Purpose

Returns the root, the parent, and the child windows of a specified window.

Protocol Format

Window: **WINDOW**
=>
Root: **WINDOW**
Parent: **WINDOW** or **None**
Children: **LISTofWINDOW**

Description

The **QueryTree** protocol request returns the root, the parent, and the child windows of a specified window. If the specified window has no parent window, it returns the value of **None**. Child windows are listed in bottom-to-top stacking order.

Fields

Window	Specifies the window ID.
Root	Specifies the root window of the specified window.
Parent	Specifies the parent window of the specified window.
Children	Specifies the child windows of the specified window.

Error Code

Window

Implementation Specifics

This protocol is part of AIXwindows Run Time Environment in AIXwindows Environment/6000.

Related Information

The **XQueryTree** subroutine.

RecolorCursor Protocol Request

Purpose

Changes the color of the cursor.

Protocol Format

Cursor: **CURSOR**
ForegroundRed, ForegroundGreen, ForegroundBlue: **CARD16**
BackgroundRed, BackgroundGreen, BackgroundBlue: **CARD16**

Description

The **RecolorCursor** protocol request changes the color of the cursor. If the cursor is currently displayed, the change is visible immediately.

Fields

Cursor	Specifies the cursor.
ForegroundRed, ForegroundGreen, ForegroundBlue	Specifies the foreground color.
BackgroundRed, BackgroundGreen, BackgroundBlue	Specifies the background color.

Error Code

Cursor

Implementation Specifics

This protocol is part of AIXwindows Run Time Environment in AIXwindows Environment/6000.

Related Information

The **XRecolorCursor** subroutine.

ReparentWindow Protocol Request

Purpose

Removes a window from its current hierarchical position and inserts it as a child window of a specified parent window.

Protocol Format

Window, Parent: **WINDOW**
X, Y: **INT16**

Description

The **ReparentWindow** protocol request removes a window from its current position in the hierarchy and inserts it as a child window of a specified parent window.

If the window is mapped, an **UnmapWindow** protocol request is performed. Then the **ReparentWindow** protocol request is processed as specified.

The x and y coordinates are relative to the parent window's origin and specify the new position of the upper-left outer corner of the window. Relative to the sibling windows, the window is placed on top in the stacking order.

A **ReparentNotify** event is then generated. If the *OverrideRedirect* field in the **ReparentNotify** event is **True**, window managers should not tamper with the window.

Finally, if the window was originally mapped, a **MapWindow** protocol request is performed.

A call to the **ReparentWindow** protocol request performs normal exposure processing on formerly obscured windows. The server may not generate exposure events for regions from the initial unmap if they are immediately obscured by the final map.

A **Match** error is generated if:

- The new parent window is not on the same screen as the old parent window.
- The new parent window is the window itself or an inferior of the window.
- The window has a **ParentRelative** background and the new parent is not the same depth as the window.

Fields

Window	Specifies the window ID.
Parent	Specifies the parent of the specified window.
X	Specifies the x coordinate.
Y	Specifies the y coordinate.

Error Codes

Match

Window

Implementation Specifics

This protocol is part of AIXwindows Run Time Environment in AIXwindows Environment/6000.

Related Information

The **XReparentWindow** subroutine.

The **ReparentNotify** event.

Enhanced X-Windows Introduction
Enhanced X-Windows Protocols Overview

RotateProperties Protocol Request

Purpose

Rotates the states of window properties.

Protocol Format

Window: **WINDOW**
Delta: **INT16**
Properties: **LISTofATOM**

Description

The **RotateProperties** protocol request rotates the states of window properties in the property list..

If the property names in the *Properties* list are numbered starting from 0 and there are `N` property names in the list, then the value associated with property name `I` is `(I + delta) mod N`, for all `I` from 0 to `N - 1`. Property states rotate by delta places around a virtual ring of property names (right for positive `delta`, left for negative `delta`.)

If `delta mod N` is nonzero, a **PropertyNotify** event is generated in the order listed for each property.

If an atom occurs more than once in the list or if 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.

Fields

Window	Specifies the window ID.
Delta	Specifies the number of places to rotate properties.
Properties	Specifies the list of properties for a window.

Error Codes

Atom

Match

Window

Implementation Specifics

This protocol is part of AIXwindows Run Time Environment in AIXwindows Environment/6000.

Related Information

The **XRotateBuffers** subroutine, **XRotateWindowProperties** subroutine.

The **PropertyNotify** event.

Enhanced X-Windows Introduction
Enhanced X-Windows Protocols Overview

SendEvent Protocol Request

Purpose

Sends an event to the specified window.

Protocol Format

Destination: **WINDOW** or **PointerWindow** or **InputFocus**
Propagate: **BOOL**
EventMask: **SETofEVENT**
Event: <normal-event-format>

Description

The **SendEvent** protocol request sends an event to the *Destination* window.

If the **PointerWindow** type is specified in the *Destination* field, the *Destination* feld is replaced with the window that currently contains the pointer. If the **InputFocus** type is specified in the *Destination* field, the *Destination* field is replaced with the focus window only if it currently contains the pointer. Otherwise, the *Destination* field is replaced with the focus window.

If the *EventMask* field 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* field is the value of **False**, then the event is sent to every client selecting on *Destination* field any of the event types in the *EventMask* field.

If *Propagate* field is the value of **True** and no clients have selected on *Destination* field any of the event types in the *EventMask* field, the *Destination* field is replaced with the closest ancestor of *Destination* field for which some client has selected a type in the *EventMask* and for which no intervening window has that type as its *DoNotPropagateMask* field. If no such window exists or if the window is an ancestor of the focus window and the **InputFocus** type 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 the *EventMask* field.

The event code must be one of the core events or one of the events defined by an extension so the X Server can correctly byte-swap the contents or a **Value** error results. The contents of the event are otherwise unaltered and unchecked by the X Server except to force on the most significant bit of the event code and to correctly set the sequence number in the event.

Active grabs are ignored by this request.

Fields

Destination	Specifies the window to receive the specified event.
Propagate	Specifies which client or clients are to receive the specified event.
EventMask	Specifies the event types.
Event	Specifies the event code.

Error Codes

Value

Window

Implementation Specifics

This protocol is part of AIXwindows Run Time Environment in AIXwindows Environment/6000.

Related Information

The **XSendEvent** subroutine.

Enhanced X-Windows Introduction
Enhanced X-Windows Protocols Overview

SetAccessControl Protocol Request

Purpose

Enables or disables the use of the access control list at connection setups.

Protocol Format

Mode: {**Enable, Disable**}

Description

The **SetAccessControl** protocol request enables or disables the use of the access control list at connection setups.

The client must reside on the same host as the server or have been granted permission by a server-dependent method to execute this request, or an **Access** error results.

Field

Mode	Specifies use of the access control list.

Error Codes

Access

Value

Implementation Specifics

This protocol is part of AIXwindows Run Time Environment in AIXwindows Environment/6000.

Related Information

The **XDisableAccessControl** subroutine, **XEnableAccessControl** subroutine, **XSetAccessControl** subroutine.

Enhanced X-Windows Introduction
Enhanced X-Windows Protocols Overview

SetClipRectangles Protocol Request

Purpose

Changes the clip-mask in the *GraphicsContext* field to a specified list of rectangles and sets the clip origin.

Protocol Format

GraphicsContext: **GCONTEXT**
ClipXOrigin, *ClipYOrigin*: **INT16**
Rectangles: **LISTofRECTANGLE**
Ordering: {**UnSorted, YSorted, YXSorted, YXBanded**}

Description

The **SetClipRectangles** protocol request changes the clip-mask defined in the specified graphics context to a specified list of rectangles and sets the clip origin. Output is clipped to remain within the specified rectangles.

Clip origin is relative to the origin of the destination drawable specified in a graphics request. Rectangle coordinates are relative to the clip origin. Unless rectangles are non-intersecting, graphics results are undefined.

An empty list for *Rectangles* field disables output, and is the opposite of specifying a value of **None** as the clip-mask member in the **CreateGC** or **ChangeGC** protocol requests.

If a client knows the ordering relations for the rectangles, they can be specified in the *Ordering* field. This may allow the X Server to operate faster. If the ordering is specified, but is incorrect, the X Server may or may not generate a **Match** error. If no error is generated, graphics results are undefined.

Fields

GraphicsContext — Specifies a graphics context.

ClipXOrigin — Specifies the x coordinate of the clip-mask origin.

ClipYOrigin — Specifies the y coordinate of the clip-mask origin.

Rectangles — Specifies a list of rectangles.

Ordering — Specifies the ordering relations of the specified list of rectangles. The *Ordering* field can be specified as the following:

UnSorted — The rectangles are in arbitrary order.

YSorted — The rectangles are nondecreasing in their y origin.

YXSorted — The rectangles are nondecreasing in their y origin; and, rectangles with an equal Y origin are nondecreasing in their x origin.

YXBanded — The rectangles are nondecreasing in their y origin; rectangles with an equal y origin are nondecreasing in their x origin; and, for every possible y scan line,

all rectangles that include that scan line have an identical y origin and y extents.

Error Codes

Alloc

GContext

Match

Value

Implementation Specifics

This protocol is part of AIXwindows Run Time Environment in AIXwindows Environment/6000.

Related Information

The **XSetClipRectangles** subroutine.

The **CreateGC** protocol request, **ChangeGC** protocol request.

Enhanced X-Windows Introduction
Enhanced X-Windows Protocols Overview

SetCloseDownMode Protocol Request

Purpose

Defines the handling of client resources at connection close.

Protocol Format

Mode: {**Destroy, RetainPermanent, RetainTemporary**}

Description

The **SetCloseDownMode** protocol request defines the handling of client resources at connection close.

The default value of the *Mode* field is the **Destroy** value.

Field

Mode	Specifies how client resources are handled at connection close.

Error Code

Value

Implementation Specifics

This protocol is part of AIXwindows Run Time Environment in AIXwindows Environment/6000.

Related Information

The **XSetCloseDownMode** subroutine.

Enhanced X-Windows Introduction
Enhanced X-Windows Protocols Overview

SetDashes Protocol Request

Purpose

Sets a dashed-line style.

Protocol Format

GraphicsContext: **GCONTEXT**
DashOffset: **CARD16**
Dashes: **LISTofCARD8**

Description

The **SetDashes** protocol request sets a dashed-line style in a specified graphics context.

A list of elements must be specified for the *Dashes* field; that is, the *Dashes* field cannot be the empty set.

If the *Dashes* field has an odd number of elements, the list is concatenated with itself to produce one with an even number of elements. The initial and alternating elements specify the even dashes; the others specify the odd dashes.

Each element specifies a dash length in pixels and must be nonzero, or a **Value** error results.

The *DashOffset* field defines the phase of the pattern, specifying how many pixels into dashes the pattern should begin in any single graphics request.

Dashing is continuous through path elements combined with a join-style, but it is reset to the value specified for the *DashOffset* field each time a cap-style is applied at the endpoint of a line.

The unit of measure for dashes is the same as in the ordinary coordinate system. When possible, dash length is measured along the slope of the line, but this implementation is only required for horizontal and vertical lines. It is suggested in other instances that length be measured along the major axis of the line. The major axis is defined as the following:

- 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.
- The y axis for all other lines.

Fields

GraphicsContext	Specifies a graphics context.
DashOffset	Defines the phase of the pattern for a dashed-line style.
Dashes	Specifies the dash length for a dashed-line style.

Error Codes

Alloc

GContext

Value

Implementation Specifics

This protocol is part of AIXwindows Run Time Environment in AIXwindows Environment/6000.

Related Information

The **XSetDashes** subroutine.

Enhanced X-Windows Introduction
Enhanced X-Windows Protocols Overview

SetFontPath Protocol Request

Purpose

Defines the directory path to search for fonts.

Protocol Format

Path: **LISTofSTRING8**

Description

The **SetFontPath** protocol request defines the directory path used to search for fonts.

There is only one search path per server, not one per client. The strings specified are intended to be searched in the order listed, although their interpretation is operating-system dependent.

Specifying an empty list for the *Path* field restores the default path defined for the X Server.

When the **SetFontPath** protocol request is executed, the X Server flushes all cached information about fonts for which there currently are no explicit resource IDs allocated.

The meaning of the error code generated by the **SetFontPath** protocol request is system-specific.

Field

Path	Specifies a directory search path.

Error Code

Value

Implementation Specifics

This protocol is part of AIXwindows Run Time Environment in AIXwindows Environment/6000.

Related Information

The **XSetFontPath** subroutine.

Enhanced X-Windows Introduction
Enhanced X-Windows Protocols Overview

SetInputFocus Protocol Request

Purpose

Changes the input focus and last-focus-change time.

Protocol Format

Focus: **Window** or **PointerRoot** or **None**
RevertTo: **{Parent, PointerRoot, None}**
Time: **TIMESTAMP** or **CurrentTime**

Description

The **SetInputFocus** protocol request 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 value specified in the *Time* field, with the **CurrentTime** replaced by the current X Server time.

If a value of **None** is specified as the *Focus* field, all keyboard events are discarded until a new focus window is set; the *RevertTo* field is inoperative.

If a window ID is specified as the *Focus* field, the specified window becomes the focus window of the keyboard. If a generated keyboard event would normally be reported to this window or one of its inferiors, the event is reproted normally. Otherwise, the event is reported with respect to the focus window.

If the **PointerRoot** type is specified as the *Focus* field, the focus window is the root window of the screen that contains the pointer at the time of each keyboard event; the *RevertTo* field is inoperative.

The **SetInputFocus** protocol request generates the **FocusIn** and **FocusOut** events.

The window specified as the *Focus* field must be viewable at the time of the request or a **Match** error results.

If the focus window becomes unviewable later, the new focus window is specified in the *RevertTo* field.

- If the **Parent** value is specified in the *RevertTo* field, the focus window reverts to the parent window or the closest viewable ancestor of the window specified as the focus window; the value of the *RevertTo* field reverts to a value of **None**.
- If the *RevertTo* field is the **PointerRoot** or **None** value, the *Focus* field reverts to that value.

When the focus window changes, the **FocusIn** and **FocusOut** events are generated, but the last-focus-change time is not affected.

Fields

Focus	Specifies the focus window.
RevertTo	Specifies the new focus window if a previously specified window becomes unviewable.
Time	Specifies the current X Server time or a timestamp expressed in milliseconds.

Error Codes

Match

Value

Window

Implementation Specifics

This protocol is part of AIXwindows Run Time Environment in AIXwindows Environment/6000.

Related Information

The **XSetInputFocus** subroutine.

Enhanced X-Windows Introduction
Enhanced X-Windows Protocols Overview

SetModifierMapping Protocol Request

Purpose

Specifies the key codes to be used as modifiers.

Protocol Format

KeycodesPerModifier: **CARD8**
Keycodes: **LISTofKEYCODE**
=>
Status: {**Success, Busy, Failed**}

Description

The **SetModifierMapping** protocol request specifies the key codes, if any, to be used as modifier keys.

The number of key codes in the list must be 8 * the *KeycodesPerModifier* field; otherwise, a **Length** error occurs.

The keycodes are divided into eight sets, with each set containing the *KeycodesPerModifier* field elements. The sets are assigned, in order, to the following modifier keys: **Shift, Lock, Control, Mod1, Mod2, Mod3, Mod4, and Mod5**.

Only nonzero key code values are used within each set; 0 values are ignored. All no-zero key codes must be in the range specified by the *min_keycode* and *max_keycode* fields at connection setup; otherwise, a **Value** error occurs.

The order of the keycodes within a set does not matter. If only 0 values are specified, the use of the corresponding modifier key is disabled, and the modifier bit have a value of 0. Otherwise, when at least one of the keys in the corresponding set is in the down position, the modifier bit will have a value of 1.

A server can impose restrictions on how modifier keys can be changed: for example, if certain keys do not generate up transitions in hardware; if the autorepeat cannot be disabled for certain keys; or, if multiple keys per modifier are not supported.

Fields

KeycodesPerModifier Specifies the maximum number of keys per modifier.

Keycodes Specifies an 8 by the *KeycodesPerModifer* field array of modifiers.

Status Returns the status of the request. If the *Status* field returns the following values:

Failed An attempt is made to override a server-dependent restriction; no modifier key is changed.

Busy New keycodes specified are nonzero and differ from those currently defined, but a modifier key (existing or new) is in the down state at the time; no modifier key is changed.

Success Modifier keys are changed and a **MappingNotify** event is generated.

Error Codes

Alloc

Length

Value

Implementation Specifics

This protocol is part of AIXwindows Run Time Environment in AIXwindows Environment/6000.

Related Information

The **XSetModifierMapping** subroutine.

Enhanced X-Windows Introduction
Enhanced X-Windows Protocols Overview

SetPointerMapping Protocol Request

Purpose

Sets the mapping of the pointer.

Protocol Format

Map: **LISTofCARD8**
=>
Status: {**Success, Busy**}

Description

The **SetPointerMapping** protocol request sets the mapping of the pointer.

The elements of the mapping list are indexed starting with a value of 1. The index is a core button number and the element of the list defines the effective number. The length of the list must match the return value of the **GetPointerMapping** protocol request, or a **Value** error results.

A 0 element disables a button. Elements are not restricted in value by the number of physical buttons, but a **Value** error results if two elements have the same nonzero value.

Fields

Map Specifies the mapping of the pointer.

Status Returns the status of the request. If the *Status* field returns the following values:

Busy A button to be changed is in the down state; mapping is not changed.

Success The mapping is changed and a **MappingNotify** event is generated.

Error Code

Value

Implementation Specifics

This protocol is part of AIXwindows Run Time Environment in AIXwindows Environment/6000.

Related Information

The **XSetPointerMapping** subroutine.

Enhanced X-Windows Introduction
Enhanced X-Windows Protocols Overview

SetScreenSaver Protocol Request

Purpose

Sets the status and method for the screen-saver.

Protocol Format

Timeout, Interval: **INT16**
PreferBlanking: {**Yes, No, Default**}
AllowExposures: {**Yes, No, Default**}

Description

The **SetScreenSaver** protocol request sets screen-saver status and method.

For each screen, if the screen blanking is preferred, the screen goes blank if this is supported by the hardware. When screen blanking is not preferred, the screen is changed in a server-dependent fashion to avoid phosphor burn, if exposures are allowed or the screen can be regenerated without sending exposure events to clients. Otherwise, the state of the screens does not change and the screen-saver function is not activated.

The screen-saver function is deactivated and all screen states are restored upon input from the keyboard or pointer, or by a **ForceScreenSaver** protocol request with a value of **Reset** for the *Mode* field.

Periodic change as a screen-saver method is server-dependent. If it is supported, the value of the *Interval* field is a hint for the frequency of screen changing. A value of 0 hints that no periodic change is requested.

Examples of ways to periodically change the screen include: scrambling the colormap; moving an icon image; or, tiling the screen with the root window background tile, randomly reorigined.

Fields

Timeout — Specifies the time before the screen-saver is activated. Values for the *Timeout* and *Interval* fields are specified in seconds. If the *Timeout* field is specified as the following:

Nonzero	The screen-saver function is enabled. Once enabled, the screen-saver function is activated if the keyboard or pointer do not receive input within the period specified by the *Timeout* field.
0	The screen-saver function is disabled.
–1	The default screen-saver status is restored. Other negative values generate a **Value** error.

Interval — Specifies the time between periodic screen-saver activity.

PreferBlanking — Specifies the type of screen blanking.

AllowExposures — Specifies exposure event reporting.

Error Code

Value

Implementation Specifics

This protocol is part of AIXwindows Run Time Environment in AIXwindows Environment/6000.

Related Information

The **XSetScreenSaver** subroutine.

Enhanced X-Windows Introduction
Enhanced X-Windows Protocols Overview

SetSelectionOwner Protocol Request

Purpose

Changes the owner, owner window, and last-change time of a specified selection.

Protocol Format

Selection: **ATOM**
Owner: **WINDOW** or **None**
Time: **TIMESTAMP** or **CurrentTime**

Description

The **SetSelectionOwner** protocol request changes the owner, owner window, and last-change time of the specified selection.

The **SetSelectionOwner** protocol request has no effect if the specified time is earlier than the last-change time of the specified selection or if it is later than the current X Server time. Otherwise, the last-change time is set to the specified time and the **CurrentTime** fype is replaced by the current X Server time.

If the **None** type is specified for the *Owner* field, the selection will have no owner. Otherwise, the owner of the selection is 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 the **None** type, the current owner is sent a **SelectionClear** event.

If a client that is the owner of a selection is later terminated (that is, its connection is closed) or if the owner window it has specified in the request is later destroyed, the owner of the selection automatically reverts to a **None** type. However, the last-change time is not affected.

This protocol has no effect if the specified *Time* field is earlier than the current last-change time of the specified *Selection* field or is later than the current server time. Otherwise, the last-change time is set to the specified *Time* field and the **CurrentTime** type is replaced by the current server time.

The X Server does not interpret the selection atom. The owner window is returned by the **GetSelectionOwner** protocol request and is reported in the **SelectionRequest** and **SelectionClear** events.

Selections are global to the X Server.

Fields

Selection	Specifies the selected atom.
Owner	Specifies the owner of the selection.
Time	Specifies the current X Server time or a timestamp expressed in milliseconds.

Error Codes

Atom

Window

Implementation Specifics

This protocol is part of AIXwindows Run Time Environment in AIXwindows Environment/6000.

Related Information

The **XSetSelectionOwner** subroutine.

Enhanced X-Windows Introduction
Enhanced X-Windows Protocols Overview

StoreColors Protocol Request

Purpose

Changes the colormap entries of specified pixels.

Protocol Format

Colormap: **COLORMAP**
Items: **LISTofCOLORITEM**

where:

COLORITEM: [*Pixel*: **CARD32**
DoRed, DoGreen, DoBlue: **BOOL**
Red, Green, Blue: **CARD16**]

Description

The **StoreColors** protocol request changes the colormap entries of specified pixels. If the colormap for the screen is an installed map, changes are visible immediately.

The *DoRed, DoGreen*, and *DoBlue* fields of each item indicate which components are to be changed.

All specified pixels in the colormap that are allocated by any client as writable are changed, even if one or more pixels produce an error.

- If a specified pixel is not a valid index into the colormap, a **Value** error is generated.
- If a specified pixel is unallocated or is allocated as read-only, an **Access** error is generated.

If more than one pixel is in error, it is arbitrary as to which pixel is reported.

Fields

Colormap	Specifies a colormap.
Items	Specifies a list of color components to be changed.
Pixel	Specifies an entry into the colormap.
DoRed, DoGreen, DoBlue	Specifies which components to change.
Red, Green, Blue	Specifies a color component.

Error Codes

Access

Colormap

Value

Implementation Specifics

This protocol is part of AIXwindows Run Time Environment in AIXwindows Environment/6000.

Related Information

The **XStoreColor** subroutine, **XStoreColors** subroutine.

Enhanced X-Windows Introduction
Enhanced X-Windows Protocols Overview

StoreNamedColor Protocol Request

Purpose

Stores a colormap entry for a specified color name.

Protocol Format

Colormap: **COLORMAP**
Pixel: **CARD32**
Name: **STRING8**
DoRed, DoGreen, DoBlue: **BOOL**

Description

The **StoreNamedColor** protocol request searches for a named color for the screen associated with a specified colormap, then stores the colormap entry for the specified pixel. The specified name should use the ISO Latin-1 encoding, and is not case-sensitive.

- If a specified pixel is not a valid index into the colormap, a **Value** error is generated.
- If a specified pixel is unallocated or is allocated as read-only, an **Access** error is generated.

Fields

Colormap	Specifies a colormap.
Pixel	Specifies a pixel.
Name	Specifies a color name.
DoRed, DoGreen, DoBlue	Specifies which component to change.

Error Codes

Access

Colormap

Name

Value

Implementation Specifics

This protocol is part of AIXwindows Run Time Environment in AIXwindows Environment/6000.

Related Information

The **XStoreNamedColor** subroutine.

Enhanced X-Windows Introduction
Enhanced X-Windows Protocols Overview

TranslateCoordinates Protocol Request

Purpose

Translates coordinate values from a specified source window to a specified destination window.

Protocol Format

SourceWindow, DestinationWindow: **WINDOW**
SourceX, SourceY: **INT16**
=>
SameScreen: **BOOL**
Child: **WINDOW** or **None**
DestinationX, DestinationY: **INT16**

Description

The **TranslateCoordinates** protocol request translates coordinate values from a specified source window to a specified destination window.

The coordinate values for the *SourceX* and *SourceY* fields are relative to the origin of the source window. These values are returned as the coordinate values of the *DestinationX* and *DestinationY* fields relative to the origin of the destination window.

If a value of **False** is returned for the *SameScreen* field, then the specified source window and destination window are on different screens, and a value of 0 is returned for the coordinate values for the *DestinationX* and *DestinationY* fields.

If the coordinates are contained in a mapped child window of the destination window, the window ID of that child window is returned in the *Child* field.

Fields

SourceWindow	Specifies the window ID of the source window.
DestinationWindow	Specifies the window ID of the destination window.
SourceX	Specifies the x coordinate relative to the source window.
SourceY	Specifies the y coordinate relative to the source window.
SameScreen	Returns the status of the screen for the specified destination window relative to the specified source window.
Child	Returns the window ID of a mapped child window of the destination window.
DestinationX	Returns the x coordinate relative to the destination window.
DestinationY	Returns the y coordinate relative to the destination window

Error Code

Window

Implementation Specifics

This protocol is part of AIXwindows Run Time Environment in AIXwindows Environment/6000.

Related Information

The **XTranslateCoordinates** subroutine.

Enhanced X-Windows Introduction
Enhanced X-Windows Protocols Overview

UngrabButton Protocol Request

Purpose

Releases the button-key combination of a passive pointer grab.

Protocol Format

Modifiers: **SETofKEYMASK** or **AnyModifier**
Button: **BUTTON** or **AnyButton**
GrabWindow: **WINDOW**

Description

The **UngrabButton** protocol request releases the button-key combination of a passive pointer grab for a specified window if it was grabbed by the specified client. It has no effect on an active grab.

If the **AnyModifier** type is specified for the *Modifiers* field, the request is issued for all possible modifier combinations, including the combination of no modifiers.

If the **AnyButton** type is specified for the *Button* field, the request is issued for all possible buttons.

Fields

Modifiers	Specifies the set of key masks.
Button	Specifies the set of pointer buttons.
GrabWindow	Specifies the window ID.

Error Code

Window

Implementation Specifics

This protocol is part of AIXwindows Run Time Environment in AIXwindows Environment/6000.

Related Information

The **XUngrabButton** subroutine.

Enhanced X-Windows Introduction
Enhanced X-Windows Protocols Overview

UngrabKey Protocol Request

Purpose

Releases the key combination for a specified window.

Protocol Format

Key: **KEYCODE** or **AnyKey**
Modifiers: **SETofKEYMASK** or **AnyModifier**
GrabWindow: **WINDOW**

Description

The **UngrabKey** protocol request releases the key combination for a specified window if it was grabbed by the specified client. It has no effect on an active grab.

If the **AnyModifier** type is specified for the *Modifiers* field, the request is issued for all possible modifier combinations, including the combination of no modifiers.

If the **AnyKey** type is specified for the *Key* field, the request is issued for all possible key codes.

Fields

Key	Specifies the set of key codes.
Modifiers	Specifies the set of key masks.
GrabWindow	Specifies the window ID.

Error Codes

Value

Window

Implementation Specifics

This protocol is part of AIXwindows Run Time Environment in AIXwindows Environment/6000.

Related Information

The **XUngrabKey** subroutine.

Enhanced X-Windows Introduction
Enhanced X-Windows Protocols Overview

UngrabKeyboard Protocol Request

Purpose

Releases the keyboard from an active keyboard grab.

Protocol Format

Time: **TIMESTAMP** or **CurrentTime**

Description

The **UngrabKeyboard** protocol request releases the keyboard if the client has grabbed the keyboard actively with the **GrabKeyboard** or the **GrabKey** protocol requests. It releases any queued events but has no effect if the specified time is earlier than the last keyboard-grab time or is later than the current X Server time.

The **UngrabKeyboard** protocol request generates the **FocusIn** and **FocusOut** events.

The **UngrabKeyboard** protocol occurs automatically if the event window for an active keyboard grab becomes unviewable.

Field

Time	Specifies the current X Server time or a timestamp expressed in milliseconds.

Implementation Specifics

This protocol is part of AIXwindows Run Time Environment in AIXwindows Environment/6000.

Related Information

The **XUngrabKeyboard** subroutine.

Enhanced X-Windows Introduction
Enhanced X-Windows Protocols Overview

UngrabPointer Protocol Request

Purpose

Releases the pointer.

Protocol Format

Time: **TIMESTAMP** or **CurrentTime**

Description

The **UngrabPointer** protocol request releases the pointer if the client has grabbed the pointer actively with the **GrabPointer** protocol request, the **GrabButton** protocol request, or a normal button press. It releases any queued events but has no effect if the specified time is earlier than the last pointer-grab time or is later than the current X Server time.

The **UngrabPointer** protocol request generates the **EnterNotify** and **LeaveNotify** events.

The **UngrabPointer** protocol request occurs automatically:

- If the event window becomes unviewable.
- If the *ConfineTo* field window of an active pointer grab request becomes unviewable.
- If window reconfiguration causes the *ConfineTo* field window of an active pointer grab request to lie completely outside the boundaries of the root window.

Fields

Time	Specifies the current X Server time or a timestamp expressed in milliseconds.

Implementation Specifics

This protocol is part of AIXwindows Run Time Environment in AIXwindows Environment/6000.

Related Information

The **XUngrabPointer** subroutine.

Enhanced X-Windows Introduction
Enhanced X-Windows Protocols Overview

UngrabServer Protocol Request

Purpose

Restarts processing of protocols and close-downs.

Description

The **UngrabServer** protocol request restarts processing of protocol requests and close-downs on other connections.

Implementation Specifics

This protocol is part of AIXwindows Run Time Environment in AIXwindows Environment/6000.

Related Information

The **XUngrabServer** subroutine.

Enhanced X-Windows Introduction
Enhanced X-Windows Protocols Overview

UninstallColormap Protocol Request

Purpose

Removes a colormap from the required list for its screen.

Protocol Format

Colormap: **COLORMAP**

Description

If a specified colormap is on the required list for its screen, the **UninstallColormap** protocol request removes it from the list. In addition, the colormap might be uninstalled, and additional colormaps might be implicitly installed or uninstalled. Whether a colormap gets installed or uninstalled is server-dependent; however, the required list must remain installed.

When a specified colormap becomes uninstalled, a **ColormapNotify** event is generated on each window where this colormap is an attribute. In addition, the **ColormapNotify** events are generated if other colormaps are installed or uninstalled as a result of this protocol request.

Field

Colormap Specifies a colormap.

Error Code

Colormap

Implementation Specifics

This protocol is part of AIXwindows Run Time Environment in AIXwindows Environment/6000.

Related Information

The **XUninstallColormap** subroutine.

The **InstallColormap** protocol request.

Enhanced X-Windows Introduction
Enhanced X-Windows Protocols Overview

UnmapSubwindows Protocol Request

Purpose

Unmaps the children windows of a specified window.

Protocol Format

Window: **WINDOW**

Description

The **UnmapSubwindows** protocol request performs an **UnmapWindow** protocol request on all mapped children windows of the specified window. Windows are unmapped in bottom-to-top stacking order.

Field

Window Specifies the window ID.

Error Code

Window

Implementation Specifics

This protocol is part of AIXwindows Run Time Environment in AIXwindows Environment/6000.

Related Information

The **XUnmapSubwindows** subroutine.

The **UnmapWindow** protocol request.

Enhanced X-Windows Introduction
Enhanced X-Windows Protocols Overview

UnmapWindow Protocol Request

Purpose

Unmaps a specified window.

Protocol Format

Window: **WINDOW**

Description

The **UnmapWindow** protocol request unmaps a specified window and generates an **UnmapNotify** event.

The **UnmapWindow** protocol request enables normal exposure processing on formerly obscured windows.

If the specified window is already unmapped, the **UnmapWindow** protocol request has no effect.

Field

Window Specifies the window ID.

Error Code

Window

Implementation Specifics

This protocol is part of AIXwindows Run Time Environment in AIXwindows Environment/6000.

Related Information

The **XUnmapWindow** subroutine.

The **UnmapSubwindows** protocol request.

Enhanced X-Windows Introduction
Enhanced X-Windows Protocols Overview

WarpPointer Protocol Request

Purpose

Changes the current position of the pointer.

Protocol Format

SourceWindow: **WINDOW** or **None**
DestinationWindow: **WINDOW** or **None**
SourceX, *SourceY*: **INT16**
SourceWidth, *SourceHeight*: **CARD16**
DestinationX, *DestinationY*: **INT16**

Description

The **WarpPointer** protocol request generates an instantaneous change in the current position of the pointer. It generates events as if the user had moved the pointer instantaneously.

If a value other than the **None** type is specified for the *SourceWindow* field, the position of the pointer changes only if the pointer is contained in the window specified and in the rectangle specified by the *SourceWidth* and *SourceHeight* field values.

If the **None** type is specified for the *DestinationWindow* field, the position of the pointer changes relative to its current position by the coordinate values specified in the *DestinationX* and *DestinationY* field.

If a window ID is specified for the *DestinationWindow* field, the position of the pointer changes relative to the origin of the specified window by the coordinate values specified in the *DestinationX* and *DestinationY* field.

The coordinates specified in the *SourceX* and *SourceY* field are relative to the origin of the window specified by the *SourceWindow* field.

If a value of 0 is specified for the *SourceWidth* field, this value is replaced with the current width of the *SourceWindow* minus *SourceX*.

If a value of 0 is specified for the *SourceHeight* field, this value is replaced with the current height of the *SourceWindow* field minus the *SourceY* field.

The **WarpPointer** protocol request cannot be used to move the pointer outside a *ConfineTo* field window of an active pointer grab. This protocol request moves the pointer only as far as the closest edge of such a window.

Fields

SourceWindow	Specifies a window ID.
DestinationWindow	Specifies a window ID.
SourceX	Specifies the X coordinate relative to the origin of the source window.
SourceY	Specifies the y coordinate relative to the origin of the source window.
SourceWidth	Specifies the width of the source rectangle.

SourceHeight	Specifies the height of the source rectangle.
DestinationX	Specifies the x coordinate for the destination of the pointer.
DestinationY	Specifies the y coordinate for the destination of the pointer.

Error Code

Window

Implementation Specifics

This protocol is part of AIXwindows Run Time Environment in AIXwindows Environment/6000.

Related Information

The **XWarpPointer** subroutine.

Enhanced X–Windows Extensions

XAIXCheckTypedWindowEvent Extension Subroutine

Purpose

Returns the next matched event in the queue for the specified window.

Library

Enhanced X-Windows Library (**libXext.a**)

Syntax

int XAIXCheckTypedWindowEvent(*DisplayPtr*, *WindowID*, *EventType*, *EventReturn*)
Display **DisplayPtr*;
Window *WindowID*;
int *EventType*;
XEvent **EventReturn*;

Description

The **XAIXCheckTypedWindowEvent** extension subroutine returns the next matched event in the queue for the specified window. First, it searches the event queue, and then any events available on the server for an event that matches the specified type and window.

When the **XAIXCheckTypedWindowEvent** extension subroutine finds a match, it removes the event from the queue, copies it into the specified **XEvent** structure and returns the value of **True**. Other events in the queue are not discarded. If the event is not available, the **XAIXCheckTypedWindowEvent** extension subroutine flushes the output buffer and returns the value of **False**.

Parameters

DisplayPtr	Specifies the connection to the X Server.
EventReturn	Specifies a client-supplied structure that receives a copy of the associated structure event that matches the extension subroutine event.
EventType	Specifies the event type to compare.
WindowID	Specifies the window ID.

Implementation Specifics

This extension subroutine is part of AIXwindows Run Time Environment in AIXwindows Environment/6000.

XAIXCheckWindowEvent Extension Subroutine

Purpose

Removes the next extension event that matches both the passed window and the passed mask.

Library

Enhanced X-Windows Library (**libXext.a**)

Syntax

Bool XAIXCheckWindowEvent(*DisplayPtr*, *WindowID*, *ExtensionEventMask*, *EventReturn*)
Display **DisplayPtr*;
Window *WindowID*;
unsigned long *ExtensionEventMask*;
XEvent **EventReturn*;

Description

The **XAIXCheckWindowEvent** extension subroutine removes the next extension event that matches both the passed window and the passed mask. This subroutine does not block and it returns the value of **0** or **1** to indicate if the event was returned.

The **XIAXCheckWindowEvent** extension subroutine first searches the event queue, and then 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, the **XAIXCheckWindowEvent** extension subroutine removes that extension event, copies it into the specified **XEvent** structure, and returns the value of **True**. The other events stored in the queue are not discarded. If an event is not found, the **XAIXCheckWindowEvent** extension subroutine flushes the output buffer and returns the value of **False**.

Parameters

DisplayPtr	Specifies the connection to the X Server.
EventReturn	Specifies a client-supplied structure that receives a copy of the associated structure event that matches the extension event.
ExtensionEventMask	Specifies the event mask, which is the bitwise-inclusive OR of one or more of the valid extension event mask bits.
WindowID	Specifies the window ID for the window with the next matched extension event to be removed.

Implementation Specifics

This extension subroutine is part of AIXwindows Run Time Environment in AIXwindows Environment/6000.

XAIXMaskEvent Extension Subroutine

Purpose

Removes the next event that matches an extension event mask.

Library

Enhanced X-Windows Library (**libXext.a**)

Syntax

XAIXMaskEvent(*DisplayPtr*, *ExtensionEventMask*, *EventReturn*)
Display **DisplayPtr*;
unsigned long *ExtensionEventMask*;
XEvent **EventReturn*;

Description

The **XAIXMaskEvent** extension subroutine removes the next event that matches an extension event mask. This extension subroutine searches the event queue for the extension events associated with the specified mask. When the **XAIXMaskEvent** extension subroutine 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, the **XAIXWindowEvent** extension subroutine flushesthe output buffer and blocks until one is received.

The **XAIXWindowEvent** extension subroutine remains blocked if extension events do not come across the wire.

Parameters

DisplayPtr	Specifies the connection to the X Server.
EventReturn	Specifies a client-supplied structure that receives a copy of the associated structure event that matches the extension event.
ExtensionEventMask	Specifies the extension event mask, which is the bitwise-inclusive OR of one or more of the valid event mask bits.

Implementation Specifics

This extension subroutine is part of AIXwindows Run Time Environment in AIXwindows Environment/6000.

Key Concepts

Purpose

Removes the next event that matches both a window and an extension event mask.

Library

Enhanced X-Windows Library (**libXext.a**)

Syntax

XAIXWindowEvent(*DisplayPtr*, *WindowID*, *ExtensionEventMask*, *EventReturn*)
Display **DisplayPtr*;
Window *WindowID*;
unsigned long *ExtensionEventMask*;
XEvent **EventReturn*;

Description

The **XAIXWindowEvent** extension subroutine removes the next event that matches both a window and an extension event mask. This extension subroutine 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, the **XAIXWindowEvent** extension subroutine flushes the output buffer and blocks until one is received.

The **XAIXWindowEvent** extension subroutine remains blocked if extension events do not come across the wire.

Parameters

DisplayPtr	Specifies the connection to the X Server.
EventReturn	Specifies a client-supplied structure that receives a copy of the associated structure event that matches the extension event.
ExtensionEventMask	Specifies the event mask, which is the bitwise-inclusive OR of one or more of the valid extension event mask bits.
WindowID	Specifies the window ID of the next matched extension event to be removed.

Implementation Specifics

This extension subroutine is part of AIXwindows Run Time Environment in AIXwindows Environment/6000.

XActivateAutoLoad Extension Subroutine

Purpose

Sets the mode of the dial or lighted programmable function key (**LPFK**) devices to the **AutoLoad** mode.

Library

Enhanced X-Windows Library (**libXext.a**)

Syntax

XActivateAutoLoad(*DisplayPtr***)**
Display ***DisplayPtr**;

Description

The **XActivateAutoLoad** extension subroutine sets the mode of the dial or **LPFK** devices to the **AutoLoad** mode.

The **LPFK** devices and dials can operate in either the **AutoLoad** or **EventReport** mode. In the **AutoLoad** mode, the X Server automatically installs the attributes to the specified devices. In the **EventReport** mode, the client is responsible for setting the appropriate attributes of the specified device.

This is analogous to the cursor automatically changing shape when crossing a window boundary, when the cursor is set in the window structure through the **XSetWindowAttributes** subroutine.

Parameter

DisplayPtr Specifies the connection to the X Server.

Implementation Specifics

This extension subroutine is part of AIXwindows Run Time Environment in AIXwindows Environment/6000.

XAsyncInput Extension Subroutine

Purpose

Sets up asynchronous input support.

Library

Enhanced X-Windows Library (**libXext.a**)

Syntax

int (***XAsyncInput**(*DisplayPtr*, *InputHandler*))()
Display **DisplayPtr*;
int (**InputHandler*)();

Description

The **XAsyncInput** extension subroutine sets up asynchronous input support. Once it is set up, the routine passed as the *InputHandler* parameter is called for each event received by the client. If the input handler returns the value of **0**, the event is discarded. If the input handler returns the value of **1**, the event is queued and another Enhanced X-Windows call can be used to remove it from the queue.

InputHandler(*DisplayPtr*, *Event*, *Level*);
Display **DisplayPtr*;
XEvent **Event*;
int *Level*;

- The *Event* parameter specifies the event from the X Server.
- The *Level* parameter specifies the level of asynchronous input.

The input handler can make Enhanced X-Windows calls with the *Level* parameter indicating the depth of nesting. If an application is on an **XNextEvent()** extension subroutine, the input handler still sees every event first.

Parameters

DisplayPtr	Specifies the connection to the X Server.
InputHandler	Specifies the procedure called for every event reported by the X Server.

Implementation Specifics

This extension subroutine is part of AIXwindows Run Time Environment in AIXwindows Environment/6000.

XBlink Extension Subroutine

Purpose

Causes the *ColormapID* parameter to be installed and the screen to blink with the specified information.

Library

Enhanced X-Windows Library (**libXext.a**)

Syntax

void XBlink (*DisplayPtr*, *ColormapID*, *Rate*, *XColorPtr*)
Display **DisplayPtr*;
Colormap *ColormapID*;
Time *Rate*;
XColor **XColorPtr*;

Description

The **XBlink** extension subroutine causes the *ColormapID* to be installed and the screen to blink with the information specified in the *XColorPtr* parameter. If the colormap is not already installed, the extension will remember the colot and blink whenever the colormap is installed with a call to the **XInstallColormap** subroutine or through the setting of the *ColormapID* attribute with the **XSetWindowAttributes** subroutine. The blinking is accomplished by swapping the **rgb** value in the colormap at the index specified by the **pixel** field in the **XColor** data structure with the **rgb** value specified in the **XColor** data structure.

The colormap entry will continue to blink at the rate specified by the *Rate* parameter until the **XBlink** extension subroutine is called with a rate of 0. If the **XBlink** extension subroutine is called multiple times with the same colormap and the same rate, all the colors will blink synchronously.

Parameters

ColormapID	Specifies colormap resource ID of the colormap to be used for the blinking.
DisplayPtr	Specifies the connection to the X Server.
Rate	Specifies number of milliseconds to wait between blinks.
XColorPtr	Specifies the pointer to color information needed to blink.

Error Code

BadAlloc

BadColor

BadColormap

Implementation Specifics

This extension subroutine is part of AIXwindows Run Time Environment in AIXwindows Environment/6000.

Related Information

The **XColor** data structure.

XCreateCrossHairCursor Extension Subroutine

Purpose

Creates a pair of cross hairs and returns the cursor resource ID associated with it.

Library

Enhanced X-Windows Library (**libXext.a**)

Syntax

Cursor XCreateCrossHairCursor (*DisplayPtr, LineWidth, LineColor, Base*)
Display **DisplayPtr;*
INT32 *LineWidth;*
XColor **LineColor;*
int *Base;*

Description

The **XCreateCrossHairCursor** extension subroutine creates a pair of cross hairs and returns the cursor resource ID associated with it. This cursor ID can then be used with the**XRecolorCursor**, **XFreeCursor** and **XDefineCursor** subroutines. The **XDefineCursor** subroutine associates the cursor resource with a WindowID.

If the *LineWidth* parameter is the value of **0**, single pixel wide lines are drawn by using the server's new line width algorithms. The color specified is used as the line color for both vertical and horizontal crossbars.

The *Base* parameter specifies if the cursor to be created is a full screen cursor or a cross hair clipped to the associated WindowID. The default value is clipped to the associated WindowID or a window based cursor.

Parameters

Base	Specifies if the cursor to be created is a full screen cursor or a cross hair clipped to the associated WindowID. The default value is clipped to the associated WindowID.
LineColor	Specifies the color of the vertical and horizontal lines.
DisplayPtr	Specifies the connection to the XServer.
LineWidth	Specifies the width of the vertical and horizontal lines of the cursor.

Error Code

BadAlloc

Return Codes

CursorID	The **XCreateCrossHairCursor** extension subroutine succeeds
0	The **XCreateCrossHairCursor** extension subroutine fails

Implementation Specifics

This extension subroutine is part of AIXwindows Run Time Environment in AIXwindows Environment/6000.

XCreateMultiColorCursor Extension Subroutine

Purpose

Creates a multicolored cursor and returns the resource ID associated with the cursor.

Library

Enhanced X-Windows Library (**libXext.a**)

Syntax

Cursor XCreateMultiColorCursor (*DisplayPtr*, *Source*, *Number*, *Colors*, *X*, *Y*)
Display **DisplayPtr*;
Pixmap *Source*;
INT32 *Number*;
XColor **Colors*;
INT32 *X*, *Y*;

Description

The **XCreateMultiColorCursor** extension subroutine creates a multicolored cursor and returns the X Server cursor resource ID associated with the cursor. This cursor ID can then be used with the **XRecolorMultiColorCursor** extension subroutine, and the **XFreeCursor** and **XDefineCursor** subroutines. The **XDefineCursor** subroutine associates the cursor resource with a WindowID.

The cursor is colored according to the colors specified in an array of **XColor** structures. The pixel value in each array element corresponds to the pixel value in the source pixmap. This pixel value in the source pixmap is colored with color specified by the **rgb** values in the same array element. Typically, the 0 value in a cursor pixmap is considered transparent.

To create a multicolored cursor there must be hardware available that contains multiple cursor planes. If hardware support is not available, a software cursor resource is created by using the color specified by array element 1 for any pixel in the pixmap that contains a pixel value. A pixel value of 0 will continue to be transparent.

Any source pixmap pixel value without a corresponding **rgb** value specified in the array of **XColor** structures is considered undefined. Any pixel value outside the range of the hardware capability is ignored.

Parameters

Colors	Specifies an array of **XColor** structures that describe the values for each pixel value in the source pixmap.
DisplayPtr	Specifies the connection to the X Server.
Number	Specifies the number of colors in the *Colors* parameter.
Source	Specifies the X Server resource ID of the pixmap to be used for the cursor.
X	Indicates the x coordinate for the cursor hotspot relative to the origin of the source pixmap. This coordinate must be a point within the source pixmap.
Y	Indicates the y coordinate for the cursor hotspot relative to the origin of the source pixmap. This coordinate must be a point within the source pixmap.

Error Codes

BadAlloc

BadDrawable

BadMatch

BadPixmap

Return Codes

CursorID	Succeeds
0	Fails

Implementation Specifics

This extension subroutine is part of AIXwindows Run Time Environment in AIXwindows Environment/6000.

Related Information

The **XColor** data structure.

XDirectAdapterAccess Extension Subroutine

Purpose

Indicates to the X Server that the client will be a direct access client.

Library

Enhanced X-Windows Library (**libXext.a**)

Syntax

#include <gai/adapter.h>

gAdapterPtr XDirectAdapterAccess (*DisplayPtr*, *screenNumber*, *argc*, *argv*)
Display **DisplayPtr*;
int *screenNumber*;
int *argc*;
char *argv*;

Description

The **XDirectAdapterAccess** extension subroutine call indicates to the X Server that the client will be a direct access client. The **XDirectAdapterAccess** extension subroutine returns a **gAdapterPtr**. The *argc* and the *argv* parameters are used to pass command line parameters and may or may not be of any use.

Parameters

argc	Specifies the argument list.
argv	Specifies the number of arguments.
DisplayPtr	Specifies the connection to the X Server.
screenNumber	Specifies the screen number.

Return Values

NULL	A **NULL** pointer will be returned if direct adapter access is not possible.
gAdapterPtr	A pointer to the adapter structure in shared memory.

Error Code

BadAlloc

Implementation Specifics

This extension subroutine is part of AIXwindows Run Time Environment in AIXwindows Environment/6000.

XDirectFontAccess Extension Subroutine

Purpose

Returns a structure that contains the shared memory key and the offset of each of the necessary structures in shared memory.

Library

Enhanced X-Windows Library (**libXext.a**)

Syntax

#include <gai/font.h>

gFontPtr *XDirectFontAccess (*DisplayPtr, XFontID*)
Display **DisplayPtr*;
Font *XFontID*;

Description

The **XDirectFontAccess** extension subroutine allows client programs to directly access font structures in shared memory. Font files are mapped into shared memory when the files are opened. The **XDirectFontAccess** extension subroutine returns a pointer to these structures in shared memory.

The X Server and client must be local to perform any of the direct access functions.

Parameters

DisplayPtr	Specifies the connection to the X Server.
XFontID	Specifies the X Server font resource ID to be accessed directly.

Return Values

NULL	Indicates that Direct Font Access is not available for this adapter.
gFontPtr	Indicates that Direct Font Access is available for this adapter.

Error Codes

BadAccess

BadFont

Implementation Specifics

This extension subroutine is part of AIXwindows Run Time Environment in AIXwindows Environment/6000.

XDirectWindowAccess Extension Subroutine

Purpose

Causes the X Server to mark the specified window structure and return a shared memory pointer.

Library

Enhanced X-Windows Library (**libXext.a**)

Syntax

#include <gai/wingeom.h>

gWingeomPtr XDirectWindowAccess (*DisplayPtr*, *WindowID*)
Display **DisplayPtr*;
Drawable *WindowID*;

Description

The **XDirectWindowAccess** extension subroutine creates a GAI window geometry resource and returns a pointer to this structure in shared memory. The only attribute useful to a client is the *gsc_handle* information which is passed back by the X Server. All other attributes are set to the value of 0.

The X Server and client must be local in order to perform any of the direct access calls.

Parameters

DisplayPtr	Specifies the connection to the X Server.
WindowID	Specifies the resource ID of the window to be accessed directly.

Return Value

NULL	Indicates that direct window access is not allowed.

Error Codes

BadDrawable

BadAlloc

Implementation Specifics

This extension subroutine is part of AIXwindows Run Time Environment in AIXwindows Environment/6000.

XDisableInputDevice Extension Subroutine

Purpose

Disables the specified input device.

Library

Enhanced X-Windows Library (**libXext.a**)

Syntax

XDisableInputDevice(*DisplayPtr*, *Device*)
Display **DisplayPtr*;
int *Device*;

Description

The **XDisableInputDevice** extension subroutine disables the specified input device so that the X Server cannot report events to it.

Parameters

Device	Specifies the input device.
DisplayPtr	Specifies the connection to the X Server.

Error Code

AIXBadDevice

Implementation Specifics

This extension subroutine is part of AIXwindows Run Time Environment in AIXwindows Environment/6000.

XDrawPolyMarker Extension Subroutine

Purpose

Draws a single marker into the specified window with extended graphics context.

Library

Enhanced X-Windows Library (**libXext.a**)

Syntax

XDrawPolyMarker(*DisplayPtr*, *WindowID*, *GraphicsContext*, *X*, *Y*)
Display **DisplayPtr*;
Window *WindowID*;
GC *GraphicsContext;*
int *X, Y;*

Description

The **XDrawPolyMarker** extension subroutine draws a single marker into the specified window with extended graphics context. This extension subroutine is not affected by the tile or stipple in the graphics context.

The **XDrawPolyMarker** extension subroutine uses the *Function, PlaneMask, Foreground, ClipXOrigin,* and *ClipYOrigin* graphics context components.

Parameters

DisplayPtr	Specifies the connection to the X Server.
GraphicsContext	Specifies the graphics context.
WindowID	Specifies the window ID.
X	Specifies the x coordinate where the marker will be drawn.
Y	Specifies the y coordinate where the marker will be drawn.

Error Codes

BadDrawable

BadGC

BadMatch

Implementation Specifics

This extension subroutine is part of AIXwindows Run Time Environment in AIXwindows Environment/6000.

XDrawPolyMarkers Extension Subroutine

Purpose

Draws multiple markers in the specified window.

Library

Enhanced X-Windows Library (**libXext.a**)

Syntax

XDrawPolyMarkers(*DisplayPtr*, *WindowID*, *GraphicsContext*, *Points*, *NumberPoints*, *Mode*)
Display **DisplayPtr*;
Window *WindowID*;
GC *GraphicsContext*;
XPoint **Points*;
int *NumberPoints*;
int *Mode*;

Description

The **XDrawPolyMarkers** extension subroutine draws multiple markers in the specified window. This extension subroutine uses the *marker* extended graphics context component.

The **XDrawPolyMarkers** extension subroutine draws multiple markers into the specified window, but is not affected by the tile or stipple in the graphics context. The location of individual polymarkers is specified by an array of **XPoint** structures. The **XDrawPolyMarkers** extension subroutine draws the markers in the order listed in the array.

The **XDrawPolyMarkers** extension subroutine uses the *function, plane_mask, foreground, clip_x_origin,* and *clip_y_origin* graphics context components.

Parameters

DisplayPtr — Specifies the connection to the X Server.

GraphicsContext — Specifies the graphics context.

Mode — Specifies the coordinate mode. This parameter indicates whether the points are relative to the window origin or to the previous point. This parameter can have the following values:

- **CoordModeOrigin** — Indicates that all points after the first are relative to the window origin. The first point is always relative to the window origin.
- **CoordModePrevious** — Indicates that all points after the first are relative to the previous point.

NumberPoints — Specifies the number of points in the array.

Points — Specifies a pointer to an array of points.

WindowID — Specifies the WindowID.

Error Codes

BadDrawable

BadGC

BadMatch

BadValue

Implementation Specifics

This extension subroutine is part of AIXwindows Run Time Environment in AIXwindows Environment/6000.

XESetCloseDisplay Extension Subroutine

Purpose

Defines a procedure to call when the **XCloseDisplay** subroutine is called.

Library

Enhanced X-Windows Library (**libXext.a**)

Syntax

int (*XESetCloseDisplay(*DisplayPtr*, *Extension*, *Procedure*))()
Display **DisplayPtr*;
int *extension*;
int (**Procedure*)();

Description

The **XESetCloseDisplay** extension subroutine defines a procedure to call when the **XCloseDisplay** subroutine is called. This subroutine returns any previously defined procedure, usually the value of **NULL**.

When the **XCloseDisplay** subroutine is called, it is called with the following syntax:

```
(*Procedure)(DisplayPtr, Codes);
Display *displayptr;
XExtCodes *codes;
```

Parameters

DisplayPtr	Specifies the connection to the X-Server.
Extension	Specifies the extension number.
Procedure	Specifies the subroutine to call when the display device is closed.

Implementation Specifics

This extension subroutine is part of AIXwindows Run Time Environment in AIXwindows Environment/6000.

Related Information

The **XCloseDisplay** subroutine.

XESetCopyGC Extension Subroutine

Purpose

Defines a procedure to call when a graphics context (**GC**) is copied.

Library

Enhanced X-Windows Library (**libXext.a**)

Syntax

int (***XESetCopyGC**(*DisplayPtr*, *Extension*, *Procedure*))()
Display **DisplayPtr*;
int *Extension*;
int (**Procedure*)();

Description

The **XESetCopyGC** extension subroutine defines a procedure to call whenever a **GC** is copied. This procedure returns any previously defined procedure, usually the value of **NULL**.

When a **GC** is copied, the subroutine is called with the following syntax:

```
(*Procedure)(DisplayPtr, GraphicsContext, Codes);
Display *DisplayPtr;
GraphicsContext GraphicsContext;
XExtCodes *Codes;
```

Parameters

DisplayPtr	Specifies the display device.
Extension	Specifies the extension number.
Procedure	Specifies the routine to call when a **GC** is copied.

Implementation Specifics

This extension subroutine is part of AIXwindows Run Time Environment in AIXwindows Environment/6000.

XESetCreateFont Extension Subroutine

Purpose

Defines a procedure to call when the **XLoadQueryFont** subroutine is called.

Library

Enhanced X-Windows Library (**libXext.a**)

Syntax

int (*XESetCreateFont(*DisplayPtr*, *Extension*, *Procedure*))()
Display **DisplayPtr*;
int *Extension*;
int (**Procedure*)();

Description

The **XESetCreateFont** extension subroutine defines a procedure to call when the **XLoadQueryFont** subroutine is called. This extension returns any previously defined procedure, usually the value of **NULL**.

When the **XLoadQueryFont** subroutine is called, it is called with the following syntax:

```
(*Procedure)(DisplayPtr, FontStruct, Codes);
Display *DisplayPtr;
XFontStructure *FontStruct;
XExtCodes *Codes;
```

Parameters

DisplayPtr	Specifies the display device.
Extension	Specifies an extension number.
Procedure	Specifies a routine to call when a font is created.

Implementation Specifics

This extension subroutine is part of AIXwindows Run Time Environment in AIXwindows Environment/6000.

Related Information

The **XLoadQueryFont** subroutine.

XESetCreateGC Extension Subroutine

Purpose

Defines a procedure to call when a new graphics context is created.

Library

Enhanced X-Windows Library (**libXext.a**)

Syntax

int (*XESetCreateGC(*DisplayPtr*, *Extension*, *Procedure*))()
Display **DisplayPtr*;
int *Extension*;
int (**Procedure*)();

Description

The **XESetCreateGC** extension subroutine defines a procedure to call when a new **GC** is created. It returns any previously defined procedure, usually the value **NULL**.

When a graphics context is created, the routine is called with the following syntax:

```
(*Procedure)(DisplayPtr, GraphicsContext, Codes);
Display *display;
GC graphics context;
XExtCodes *codes;
```

Parameters

DisplayPtr	Specifies the display device.
Extension	Specifies the extension number.
Procedure	Specifies the routine to call when a graphics context is created.

Implementation Specifics

This extension subroutine is part of AIXwindows Run Time Environment in AIXwindows Environment/6000.

XESetError Extension Subroutine

Purpose

Suppresses the call to an external error handling routine and defines an alternative routine for error handling.

Library

Enhanced X-Windows Library (**libXext.a**)

Syntax

int (*XESetError(*DisplayPtr*, *Extension*, *Procedure*))()
**Display ****DisplayPtr*;
int *Extension*;
int (**Procedure*)();

Description

The **XESetError** extension subroutine suppresses the call to an external error handling routine and defines an alternative routine for error handling. This extension subroutine 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 the **Xlib** library detects a protocol error in the **_XReply** extension subroutine, it calls the procedure with the following syntax:

```
int (*Procedure)(DisplayPtr, Error, Codes, ReturnCode);
Display *DisplayPtr;
xError *Error;
XExtCodes *Codes;
int *ReturnCode;
```

- The *Error* parameter is a pointer to the 32-byte wire format error.
- The *Codes* parameter is a pointer to the extension subroutine codes structure.
- The *ReturnCode* parameter is the return code returned by the **_XReply** extension subroutine.

If the extension subroutine returns the value of zero, the error is not suppressed, and the **XError** extension subroutine is called. If the extension subroutine returns a nonzero, the error is suppressed and the **_XReply** extension subroutine returns the value in the *ReturnCode* parameter.

Parameters

DisplayPtr	Specifies a display device.
Extension	Specifies an extension number.
Procedure	Specifies a routine to call when an error code is received.

Implementation Specifics

This extension subroutine is part of AIXwindows Run Time Environment in AIXwindows Environment/6000.

XESetErrorString Extension Subroutine

Purpose

Defines a procedure to call when an I/O error is detected.

Library

Enhanced X-Windows Library (**libXext.a**)

Syntax

char ***(*XESetErrorString**(*DisplayPtr*, *Extension*, *Procedure*))()**
Display **DisplayPtr*;
int *Extension*;
char ***(**Procedure*)();

Description

The **XESetErrorString** extension subroutine defines a procedure to call when an I/O error is detected.

Parameters

DisplayPtr	Specifies a display device.
Extension	Specifies an extension number.
Procedure	Specifies a routine to call when an I/O error occurs.

Implementation Specifics

This extension subroutine is part of AIXwindows Run Time Environment in AIXwindows Environment/6000.

XESetEventToWire Extension Subroutine

Purpose

Defines a procedure to call when an event needs to be converted from the host format to the wire format.

Library

Enhanced X-Windows Library (**libXext.a**)

Syntax

int (***XESetEventToWire**(*DisplayPtr*, *EventNumber*, *Procedure*))()
Display **DisplayPtr*;
int *EventNumber*;
int (**Procedure*)();

Description

The **XESetEventToWire** extension subroutine defines a procedure to call when an event needs to be converted from the host format (the **XEvent** structure found in the **<X11/Xlib.h>** file) to the wire format (the **xEvent** structure found in the **<X11/Xproto.h>** header file). This extension returns any previously defined procedure.

Note: The host event structure size cannot be larger than the size of the **XEvent** union of structures.

When the **Xlib** library needs to convert an event from the host format to the wire format, the routine is called with the following syntax:

```
(*Procedure)(DisplayPtr, Re, Event);
Display *DisplayPtr;
XEvent *Re;
xEvent *Event;
```

- The *Re* variable is a pointer to the host format event.
- The *Event* variable is a pointer to where the 32-byte wire event structure should be stored.

In the **XEvent** structure, the *type* field should be the first element and the *window* field should be the second element. The *type* field should be copied from the **xEvent** structure. The other elements should be copied from the host format to the **XEvent** structure.

Parameters

DisplayPtr	Specifies a display device.
EventNumber	Specifies a protocol event number to replace with the conversion routine.
Procedure	Specifies a routine to call when converting an event.

Implementation Specifics

This extension subroutine is part of AIXwindows Run Time Environment in AIXwindows Environment/6000.

XESetFlushGC Extension Subroutine

Purpose

Defines a procedure to call when a graphics context needs to be updated in the server.

Library

Enhanced X-Windows Library (**libXext.a**)

Syntax

int (***XESetFlushGC**(*DisplayPtr*, *Extension*, *Procedure*))()
Display **DisplayPtr*;
int *Extension*;
char *(**Procedure*)();

Description

The **XESetFlushGC** extension subroutine defines a procedure to call when a graphics context in the cache needs to be updated in the server.

Parameters

DisplayPtr	Specifies a display device.
Extension	Specifies an extension number.
Procedure	Specifies a routine to call when the **GC** cache needs to be updated.

Implementation Specifics

This extension subroutine is part of AIXwindows Run Time Environment in AIXwindows Environment/6000.

XESetFreeFont Extension Subroutine

Purpose

Defines a procedure to call when the **XFreeFont** subroutine is called.

Library

Enhanced X-Windows Library (**libXext.a**)

Syntax

int ***XESetFreeFont**(*DisplayPtr*, *Extension*, *Procedure*))()
Display **DisplayPtr*;
int *Extension*;
int (**Procedure*)();

Description

The **XESetFreeFont** extension subroutine defines a procedure to call when the **XFreeFont** subroutine is called. This extension subroutine returns any previously defined procedure, usually the value of **NULL**.

When the **XFreeFont** subroutine is called, the defined procedure is called with the following syntax:

```
(*Procedure)(DisplayPtr, FontStructure, Codes);
Display *Display;
XFontStruct *FontStructure;
XExtCodes *Codes;
```

Parameters

DisplayPtr	Specifies a display device.
Extension	Specifies an extension number.
Procedure	Specifies a routine to call when a font is freed.

Implementation Specifics

This extension subroutine is part of AIXwindows Run Time Environment in AIXwindows Environment/6000.

XESetFreeGC Extension Subroutine

Purpose

Defines a procedure to call when a graphics context is freed.

Library

Enhanced X-Windows Library (**libXext.a**)

Syntax

int (*XESetFreeGC(*Display*, *Extension*, *Procedure*))()
**Display ****DisplayPtr*;
int *Extension*;
int (**Procedure*)();

Description

The **XESetFreeGC** extension subroutine defines a procedure to call when a graphics context is freed. This extension subroutine returns any previously defined procedure, usually the value of **NULL**.

When a graphics context is freed, the defined procedure is called with the following syntax:

```
(*Procedure)(Display, GraphicsContext, Codes)
Display *DisplayPtr;
GC GraphicsContext;
XExtCodes *Codes;
```

Parameters

DisplayPtr	Specifies the display device.
Extension	Specifies the extension number.
Procedure	Specifies the routine to call when a graphics context is freed.

Implementation Specifics

This extension subroutine is part of AIXwindows Run Time Environment in AIXwindows Environment/6000.

XESetWireToEvent Extension Subroutine

Purpose

Defines a procedure to call when an event is to be converted from wire format to host format.

Library

Enhanced X-Windows Library (**libXext.a**)

Syntax

int (***XESetWireToEvent**(*DisplayPtr*, *EventNumber*, *Procedure*))()
Display **DisplayPtr*;
int *EventNumber*;
Bool (**Procedure*)();

Description

The **XESetWireToEvent** extension subroutine defines a procedure to call when an event is to be converted from wire format (the **xEvent** structure in the **<X11/Xproto.h>** header file) to host format (the **XEvent** structure in the **<X11/Xlib.h>** header file).

The **XESetWireToEvent** extension subroutine returns any previously defined procedure.

Note: The host event structure size cannot be bigger than the size of the **XEvent** union of structures.

When the **Xlib** library needs to convert an event from wire format to natural host format, the **XESetWireToEvent** extension subroutine is called with the following syntax:

```
Status(*Procedure)(DisplayPtr, Re, Event);
Display *DisplayPtr;
XEvent *Re;
xEvent Event;
```

- The *Re* parameter is a pointer to where the host format event should be stored. It is the source (the information to be converted).
- The *Event* parameter is the 32-byte wire event structure. It is the destination (the structure that needs to be filled).

In the **XEvent** structure, the *type* field must be the first field and the *window* field 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).

Parameters

DisplayPtr	Specifies a display device.
EventNumber	Specifies a protocol event routine to replace with the conversion routine.
Procedure	Specifies a routine to call when converting the event.

Implementation Specifics

This extension subroutine is part of AIXwindows Run Time Environment in AIXwindows Environment/6000.

XEnableInputDevice Extension Subroutine

Purpose

Enables event input.

Library

Enhanced X-Windows Library (**libXext.a**)

Syntax

int XEnableInputDevice(*DisplayPtr*, *Device***)**
Display ****Display*;**
int *Device*;

Description

The **XEnableInputDevice** extension subroutine enables event input by allowing the X Server to report events from the specified input device.

Parameters

Device	Specifies the input device.
DisplayPtr	Specifies the connection to the X Server.

Error Code

AIXBadDevice

Implementation Specifics

This extension subroutine is part of AIXwindows Run Time Environment in AIXwindows Environment/6000.

XFreeExtensionList Extension Subroutine

Purpose

Frees the memory allocated by the **XListExtensions** extension subroutine.

Library

Enhanced X-Windows Library (**libXext.a**)

Syntax

XFreeExtensionList(*List*)
char ***List*;

Description

The **XFreeExtensionList** extension subroutine frees the memory allocated by the **XListExtensions** extension subroutine.

Parameter

List Specifies the allocated memory to be freed.

Implementation Specifics

This extension subroutine is part of AIXwindows Run Time Environment in AIXwindows Environment/6000.

XGetDeviceInputFocus Extension Subroutine

Purpose

Returns the current dial or lighted programmable function key (**LPFK**) input focus state and focus WindowID.

Library

Enhanced X-Windows Library (**libXext.a**)

Syntax

XGetDeviceInputFocus(*DisplayPtr*, *Device*, *FocusReturn*, *RevertToReturn*)
Display **DisplayPtr*;
int *Device*;
Window **FocusReturn*;
int **RevertToReturn*;

Description

The **XGetDeviceInputFocus** extension subroutine returns the current dial or **LPFK** input focus state and focus WindowID.

Parameters

Device	Specifies the input device. This parameter can have the following values: **DEVDIAL** **DEVLPFK**
DisplayPtr	Specifies the connection to the X-Server.
FocusReturn	Returns the focus WindowID. This parameter can have the following values: **None** **PointerRoot**
RevertToReturn	Returns the current focus state. This parameter can have the following values: **RevertToNone** **RevertToParent** **RevertToPointerRoot**

Implementation Specifics

This extension subroutine is part of AIXwindows Run Time Environment in AIXwindows Environment/6000.

Related Information

The **XGetInputFocus** subroutine.

XGetDialAttributes Extension Subroutine

Purpose

Returns the dial resolutions specified on the *DialMask* parameter of the specified window.

Library

Enhanced X-Windows Library (**libXext.a**)

Syntax

XGetDialAttributes(*DisplayPtr*, *WindowID*, *DialMask*, *Resolution*, *Number*)
Display **DisplayPtr*;
Window *WindowID*;
unsigned long *DialMask*;
char **Resolution*;
int *Number*;

Description

The **XGetDialAttributes** extension subroutine returns the dial resolutions for the dials specified by the *DialMask* parameter of the specified window. Each bit of the *DialMask* parameter represents one possible dial. The resolution array should be as large as the largest numbered dial plus 1.

For example, if the client program wanted to get the attributes for dials 0, 2, 5, and 7, it would specify the following in the *DialMask* parameter:

```
DialMask0 | DialMask2 | DialMask5 | DialMask7
```

In addition, it would create the resolution array with 8 bytes.

When the **XGetDialAttributes** subroutine is completed, it returns data in bytes 0, 2, 5, and 7 of the resolution array. Each of these bytes contains a value between 2 and 8. These values can be interpreted as follows:

Value in Array	Points Per Revolution
2	4
3	8
4	16
5	32
6	64
7	128
8	256

Parameters

DialMask	Specifies the dial mask, which is the bitwise-inclusive OR of one or more valid dial mask bits. Valid values for this parameter are the **DialMask0** through **DialMask23** values, and the **AllDialsMask** value.
DisplayPtr	Specifies the connection to the X-Server.
NumberID	Specifies the number of entries in the resolution array.
Resolution	Specifies an array of dial resolutions.
Window	Specifies the WindowID.

Implementation Specifics

This extension subroutine is part of AIXwindows Run Time Environment in AIXwindows Environment/6000.

XGetDialControl Extension Subroutine

Purpose

Returns the current dial resolutions for the dials specified by the *DialMask* parameter.

Library

Enhanced X-Windows Library (**libXext.a**)

Syntax

XGetDialControl(*DisplayPtr*, *Dialmask*, *Resolution*, *Number*)
Display **DisplayPtr*;
int *DialMask*;
char **Resolution*;
int *Number*;

Description

The **XGetDialControl** extension subroutine returns the current dial resolutions for the dials specified by the *DialMask* parameter. Each bit of the *DialMask* parameter representsone possible dial. When the resolution array is allocated, 1 byte should be allocated for each dial.

Parameters

Dialmask	Specifies the dial mask bits. Valid values for this parameter are the **DialMask0** through **DialMask23** values, and **AllDialsMask** value.
DisplayPtr	Specifies the connection to the X Server.
Number	Specifies the number of entries in the resolution array.
Resolution	Returns dial resolutions.

Implementation Specifics

This extension subroutine is part of AIXwindows Run Time Environment in AIXwindows Environment/6000.

XGetLpfkAttributes Extension Subroutine

Purpose

Retrieves the current setting for the individual keys of the lighted programmable function key (**LPFK**) device for the specified window.

Library

Enhanced X-Windows Library (**libXext.a**)

Syntax

XGetLpfkAttributes(*DisplayPtr*, *WindowID*, *LPFKMask*, *LightMask*)
Display **DisplayPtr*;
Window *WindowID*;
unsigned long **LPFKMask*;
unsigned long *LightMask*;

Description

The **XGetLpfkAttributes** extension subroutine retrieves the current setting for the individual keys of the **LPFK** device for the specified window.

Each bit in the *LPFKMask* and *LightMask* parameters represents one key. If a bit is set in the *LPFKMask* parameter, the corresponding key is enabled for input. If the bit is not set, the X-Server will not return any key press events for this key. If a bit is set in the *LightMask* parameter, the key is lighted. If it is not set, the key is not lighted.

Parameters

DisplayPtr Specifies the connection to the X-Server.

LightMask Returns the **LPFK** mask, which is the bitwise-inclusive OR of one or more valid **LPFK** mask bits. Valid values for this parameter are the **LPFKMask0** value, and the **LPFKMask2** through **LPFKMask31** values.

LPFKMask Returns the **LPFK** mask, which is the bitwise-inclusive OR of one or more valid **LPFK** mask bits. Valid values for this parameter are the **LPFKMask0** value, and the **LPFKMask2** through **LPFKMask31** values.

WindowID Specifies the WindowID.

Implementation Specifics

This extension subroutine is part of AIXwindows Run Time Environment in AIXwindows Environment/6000.

Related Information

The lighted programmable function key.

XGetLpfkControl Extension Subroutine

Purpose

Retrieves the current settings for the individual keys of the lighted programmable function key (**LPFK**) device.

Library

Enhanced X-Windows Library (**libXext.a**)

Syntax

XGetLpfkControl(*DisplayPtr*, *LPFKMask*, *LightMask***)**
**Display ****DisplayPtr*;
**unsigned long ****LPFKMask*;
**unsigned long ****LightMask*;

Description

The **XGetLpfkControl** extension subroutine retrieves the current settings for the individual keys of the lighted programmable function key (**LPFK**) device.

Each bit in the *LPFKMask* and *LightMask* parameters represents one key. If a bit is set in the *LPFKMask* parameter, the corresponding key is enabled for input. If the bit is not set, the X Server will not return any key press events for this key. If a bit is set in the *LightMask* parameter, the key is lighted. If the bit is not set, the key is not lighted.

Parameters

DisplayPtr Specifies the connection to the X Server.

LightMask Returns the current **LPFK** output mask setting. Valid values for this parameter are the **LPFKMask0** value, and the **LPFKMask1** through **LPFKMask31** values.

LPFKMask Returns the current **LPFK** input mask setting. Valid values for this parameter are the **LPFKMask0** value, and the **LPFKMask1** through **LPFKMask31** values.

Implementation Specifics

This extension subroutine is part of AIXwindows Run Time Environment in AIXwindows Environment/6000.

XListExtensions Extension Subroutine

Purpose

Returns a list of all extensions supported by the server.

Library

Enhanced X-Windows Library (**libXext.a**)

C Syntax

char **XListExtensions(*DisplayPtr, NumberExtensions***)**
**Display ****DisplayPtr*;
**int ****NumberExtensions*;

Description

The **XListExtensions** extension subroutine returns a list of all extension subroutines supported by the server.

Parameters

DisplayPtr	Specifies the display device.
NumberExtensions	Returns the number of extension subroutines.

Return Values

A list of all extension subroutines supported by the server.

Implementation Specifics

This extension subroutine is part of AIXwindows Run Time Environment in AIXwindows Environment/6000.

Related Information

The **ListExtensions** protocol request.

XListInputDevices Extension Subroutine

Purpose

Obtains a list of devices currently supported by the X Server.

Library

Enhanced X-Windows Library (**libXext.a**)

Syntax

AIXInputDeviceInfo *XListInputDevices(*DisplayPtr, NumberDevices, Enabled***)**
**Display ***DisplayPtr*;
**int ***NumberDevices*;
**Bool ***Enabled*;

Description

The **XListInputDevices** extension subroutine obtains a list of devices currently supportedby the X Server. This extension subroutine finds out which devices are available and returns a pointer to a list of device structures that were allocated by the routine.

Use the **XFree** subroutine to free the memory after this function has completed.

The **AIXInputDeviceInfo** data structure is the following:

```
Typedef struct {
   short deviceID;
    short state;
    int size;
    char *devinfo;
} AIXInputDeviceInfo;
```

deviceID Specifies the device ID, which is defined in the **<X11/XAIX.h>** file.

devinfo Specifies a pointer to a device private structure. This could be the value of **NULL**.

size Specifies the size of the *devinfo* parameter.

state Specifies the on or off state of the specified device.

Parameters

DisplayPtr Specifies the connection to the X Server.

Enabled Returns the number of currently enabled devices.

NumberDevices Returns the number of devices currently supported by the X Server.

Implementation Specifics

This extension subroutine is part of AIXwindows Run Time Environment in AIXwindows Environment/6000.

Related Information

The **XFree** subroutine.

XMaxRequestSize Extension Subroutine

Purpose

Returns the maximum request size supported by the server.

Library

Enhanced X-Windows Library (**libXext.a**)

Syntax

long XMaxRequestSize(*DisplayPtr*)
Display **DisplayPtr*

Description

The **XMaxRequestSize** extension subroutine returns the maximum request size (4-byte units) supported by the server. Single protocol requests to the server cannot be any longer than this. Extension subroutines should be designed so that long protocol requestscan be split up into smaller requests. The protocol guarantees the maximum request size to be no smaller than 4096 units (16384 bytes).

Parameter

DisplayPtr Specifies the connection to the X Server.

Implementation Specifics

This extension subroutine is part of AIXwindows Run Time Environment in AIXwindows Environment/6000.

XQueryAutoLoad Extension Subroutine

Purpose

Returns the current event mode of the dial and the lighted programmable function key (**LPFK**) devices.

Library

Enhanced X-Windows Library (**libXext.a**)

Syntax

XQueryAutoLoad (*DisplayPtr*, *OnOff*)
Display **DisplayPtr*;
int **OnOff*;

Description

The **XQueryAutoLoad** extension subroutine returns the current event mode of the dial and the **LPFK** devices.

The **LPFK** devices and dials can operate in either the **AutoLoad** or **EventReport** mode. In the **AutoLoad** mode, the X Server automatically installs the attributes to the specified devices. In the **EventReport** mode, the client is responsible for setting the appropriate attributes of the specified device.

This is analogous to the cursor automatically changing shape when crossing a window boundary, when the cursor is set in the window structure through the **XSetWindowAttributes** subroutine.

Parameters

DisplayPtr Specifies the connection to the X Server.

OnOff Returns the event mode of the dial and **LPFK** device. This parameter can have the following values:

AIXDeviceAutoLoadOff Indicates that the **EventReport** mode is on.

AIXDeviceAutoLoadOn Indicates that the **AutoLoad** mode is on.

These define values can be found in the **<AIX.h/include>** file.

Implementation Specifics

This extension subroutine is part of AIXwindows Run Time Environment in AIXwindows Environment/6000.

XQueryCrossHairCursor Extension Subroutine

Purpose

Returns information about the size and colors in a cross hair cursor for a particular display device.

Library

Enhanced X-Windows Library (**libXext.a**)

Syntax

Status XQueryCrosshairCursor (*DisplayPtr, MinWidth, MaxWidth, BestWidth, Colors, Base*)

Display **DisplayPtr*;
int **MinWidth*;
int **MaxWidth*;
int **BestWidth*;
int **Colors*;
int **Base*

Description

The **XQueryCrossHairCursor** extension subroutine returns the minimum width of the vertical and horizontal lines, the maximum width, the best width of a cross hair cursor, andthe number of colors in a cross hair cursor for a particular display device.

Parameters

BestWidth	Returns the best width of the vertical and horizontal cross hairs.
Colors	Returns the number of colors the hardware will support for a cross hair cursor.
DisplayPtr	Specifies the connection to the X Server.
MaxWidth	Returns the maximum width of the vertical and horizontal cross hairs.
MinWidth	Returns the minimum width of the vertical and horizontal cross hairs.
Base	Returns whether the base is the screen base or the window base.

Return Values

True	Succeeds
False	Fails

Implementation Specifics

This extension subroutine is part of AIXwindows Run Time Environment in AIXwindows Environment/6000.

XQueryExtension Extension Subroutine

Purpose

Determines if the named extension subroutine is present.

Library

Enhanced X-Windows LIbrary (**libXext.a**)

Syntax

Bool XQueryExtension(*DisplayPtr, Name, MajorOpCode, FirstEvent, FirstError*)
Display **DisplayPtr*;
char **Name;*
int **MajorOpCode*;
int **FirstEvent*;
int **FirstError*;

Description

The **XQueryExtension** extension subroutine determines if the named extension subroutine 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 subroutine.

Parameters

DisplayPtr	Specifies the display device.
FirstError	Specifies the returned base error code for the named extension subroutine or the value of **0**.
FirstEvent	Specifies the returned base event type code for the named extension subroutine or the value of **0**.
MajorOpcode	Specifies the returned major operation code for the named extension subroutine or the value of **0**.
Name	Specifies the name of an extension subroutine in the form of a case-sensitive ASCII string.

Implementation Specifics

This extension subroutine is part of AIXwindows Run Time Environment in AIXwindows Environment/6000.

XQueryInputDevice Extension Subroutine

Purpose

Returns the current status of the specified device.

Library

Enhanced X-Windows Library (**libXext.a**)

Syntax

XQueryInputDevice (*DisplayPtr*, *Device*, *OnOff*)
Display **DisplayPtr*;
int *Device*;
int **OnOff*;

Description

The **XQueryInputDevice** extension subroutine returns the current status of the specified device.

Parameters

Device Specifies the device ID.

DisplayPtr Specifies the connection to the X-Server.

OnOff Returns the on or off status of the specified device. This parameter can have the following values:

False Indicates that the specified device is off.

True Indicates that the specified device is on.

Implementation Specifics

This extension subroutine is part of AIXwindows Run Time Environment in AIXwindows Environment/6000.

XRecolorMultiColorCursor Extension Subroutine

Purpose

Changes one or more of the colors in a multicolored cursor.

Library

Enhanced X-Windows Library (**libXext.a**)

Syntax

Status XRecolorMultiColorCursor (*DisplayPtr*, *CursorResourceID*, *Number*, *Colors*)
Display **DisplayPtr*;
Cursor *CursorResourceID*;
INT32 *Number*;
XColor **Colors*;

Description

The **XRecolorMultiColorCursor** extension subroutine changes one or more of the colors in a multicolored cursor. The *Number* parameter indicates the number of colors to be changed. The array of **XColor** structures specifies which colors are to be changed and what their new values are.

The pixel value in each array element corresponds to the pixel value in the cursor. This pixel value will be colored with the color specified by the **rgb** values in the same array element. Typically, a pixel value of **0x00** in a cursor is considered transparent.

Parameters

Colors	Specifies the array of **XColor** structures that indicate the pixel values to be changed to new **rgb** values for these pixels.
CursorResourceID	Specifies the X Server resource ID of the cursor to be recolored used for the cursor.
DisplayPtr	Specifies the connection to the X Server.
Number	Specifies the number of colors in the *Colors* parameter.

Error Codes

BadCursor

BadValue

Return Values

True	The **XRecolorMultiColorCursor** extension subroutine succeeds.
False	The **XRecolorMultiColorCursor** extension subroutine fails.

Implementation Specifics

This extension subroutine is part of AIXwindows Run Time Environment in AIXwindows Environment/6000.

Related Information

The **XColor** data structure.

XSelectDeviceInput Extension Subroutine

Purpose

Requests the X-Server to report the events associated with the event masks from the specified device.

Library

Enhanced X-Windows Library (**libXext.a**)

Syntax

XSelectDeviceInput(*DisplayPtr*, *Device*, *WindowID*, *ExtensionEventMask*)
Display **DisplayPtr*;
int *Device*;
Window *WindowID*;
unsigned long *ExtensionEventMask*;

Description

The **XSelectDeviceInput** extension subroutine requests the X-Server to report the events associated with the event masks from the specified device.

With this extension subroutine, events are reported relative to a window. If a window is not interested in an event, the events are reported to the closest ancestor window that is interested.

A call to the **XSelectDeviceInput** extension subroutine overrides any previous call to the **XSelectDeviceInput** extension subroutine 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.

Parameters

Device — Specifies the device ID.

DisplayPtr — Specifies the connection to the X-Server.

ExtensionEventMask — Specifies the extension event mask, which is the bitwise-inclusive OR of one or more of the valid extension event mask bits. Valid values for this parameter are the following:

LPFKPressMask	**DialRotateMask**
AIXDeviceMapChangeMask	**AIXFocusChangeMask**
AIXButtonPressMask	**AIXButtonReleaseMask**
AIXKeyPressMask	**AIXKeyReleaseMask**

WindowID — Specifies the WindowID. Client applications interested in an extension event for a particular window pass that WindowID.

BadImplementation

BadValue

BadWindow

Implementation Specifics

This extension subroutine is part of AIXwindows Run Time Environment in AIXwindows Environment/6000.

Related Information

The **XSelectDial** extension subroutine.

The **XSelectInput** subroutine.

XSelectDialInput Extension Subroutine

Purpose

Requests the server to report events associated with the event masks.

Library

Enhanced X-Windows Library (**libXext.a**)

Syntax

XSelectDialInput(*DisplayPtr*, *WindowID*, *ExtensionEventMask*)
Display **DisplayPtr*;
Window *WindowID*;
unsigned long *ExtensionEventMask*;

Description

The **XSelectDialInput** extension subroutine requests the server to report events associated with the event masks passed to the *ExtensionEventMask* parameter.

With this extension subroutine, events are reported relative to a window. If a window is not interested in an event, the events are reported to the closest ancestor window that is interested.

A call to the **XSelectDialInput** extension subroutine overrides any previous call to the **XSelectDialInput** extension subroutine 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.

Parameters

DisplayPtr	Specifies the connection to the X-Server.
ExtensionEventMask	Specifies the extension event mask, which is the bitwise-inclusive OR of one or more of the valid extension event mask bits. Valid values for this parameter are the following: **AIXDeviceMapChangeMask** **DialRotateMask** **AIXButtonPressMask** **AIXButtonReleaseMask** **AIXFocusChangeMask**
WindowID	Specifies the WindowID. Client applications interested in an extension event for a particular window pass that WindowID.

Implementation Specifics

This extension subroutine is part of AIXwindows Run Time Environment in AIXwindows Environment/6000.

XSelectDial Extension Subroutine

Purpose

Associates a dial with a WindowID.

Library

Enhanced X-Windows Library (**libXext.a**)

Syntax

XSelectDial(*DisplayPtr*, *WindowID*, *DialMask*)
Display **DisplayPtr*;
Window *WindowID*;
unsigned long *DialMask*;

Description

The **XSelectDial** extension subroutine associates a dial with a WindowID. Typically, the application would call the **XSelectDialInput** extension subroutine to select the events in which it is interested, and then the **XSelectDial** extension subroutine to select the specific dials in which it is interested.

Parameters

DialMask Specifies the dial mask, which is the bitwise-inclusive OR of one or more valid dial mask bits. Valid values for this parameter are the **DialMask0** value,and the **DialMask1** through **DialMask23** values.

DisplayPtr Specifies the connection to the X Server.

WindowID Specifies the WindowID. Client applications interested in an extension event for a particular window pass that WindowID.

Error Codes

BadImplementation

BadValue

BadWindow

Implementation Specifics

This extension subroutine is part of AIXwindows Run Time Environment in AIXwindows Environment/6000.

Related Information

The **XSelectDeviceInput** extension subroutine, **XSelectDialInput** extension subroutine.

The **XSelectInput** subroutine.

XSelectLpfk Extension Subroutine

Purpose

Selects specified keys from the lighted programmable function key (**LPFK**) device for input and output for a specified window.

Library

Enhanced X-Windows Library (**libXext.a**)

Syntax

Description

XSelectLpfk(*DisplayPtr*, *WindowID*, *LPFKMask*, *LightMask*)
Display **DisplayPtr*;
Window *WindowID*;
unsigned long *LPFKMask*;
unsigned long *LightMask*;

The **XSelectLpfk** extension subroutine selects specified keys from the **LPFK** device for input and output for a specified window. Typically, the **XSelectLpfk** extension subroutine follows the **XSelectLpfkInput** extension subroutine.

Parameters

DisplayPtr Specifies the connection to the X-Server.

LightMask Allows the application to select the keys it is interested in for output. Valid values for this parameter are the **LPFKMask0** value, and the **LPFKMask1** through **LPFKMask31**.

LPFKMask Allows the application to select the keys it is interested in for input. Valid values for this parameter are the **LPFKMask0** value, and the **LPFKMask1** through **LPFKMask31**.

WindowID Specifies the WindowID. Client applications interested in an extension event for a particular window pass that WindowID.

Implementation Specifics

This extension subroutine is part of AIXwindows Run Time Environment in AIXwindows Environment/6000.

XSelectLpfkInput Extension Subroutine

Purpose

Requests the server to report events associated with the event masks for the lighted programmable function key (**LPFK**) device.

Library

Enhanced X-Windows Library (**libXext.a**)

Syntax

XSelectLpfkInput(*DisplayPtr*, *WindowID*, *ExtensionEventMask*)
Display **DisplayPtr*;
Window *WindowID*;
unsigned long *ExtensionEventMask*;

Description

The **XSelectLpfkInput** extension subroutine requests the server to report events associated with the event masks for **LPFK** device.

With this subroutine, events are reported relative to a window. If a window is not interested in an event, the events are reported to the closest ancestor window that is interested.

A call to the **XSelectLpfkInput** extension subroutine overrides any previous call to the **XSelectLpfkInput** extension subroutine 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.

Parameters

DisplayPtr Specifies the connection to the X Server.

ExtensionEventMask Specifies the extension event mask, which is the bitwise-inclusive OR of one or more of the valid extension event mask bits. Valid values for this parameter are the following:

AIXDeviceMapChangeMask	**LPFKPressMask**
AIXFocusChangeMask	**AIXButtonPressMask**
AIXButtonReleaseMask	**AIXKeyPressMask**
AIXKeyReleaseMask	

WindowID Specifies the window ID. Client applications interested in an extension event for a particular window pass that window ID.

Implementation Specifics

This extension subroutine is part of AIXwindows Run Time Environment in AIXwindows Environment/6000.

XSetDeviceInputFocus Extension Subroutine

Purpose

Sets the device input focus and the last focus–change time.

Library

Enhanced X-Windows Library (**libXext.a**)

Syntax

XSetDeviceInputFocus(*DisplayPtr*, *Device*, *Focus*, *RevertTo*, *Time*)
Display **DisplayPtr*;
int *Device*;
Window *Focus*;
int *RevertTo*;
Time *Time*;

Description

The **XSetDeviceInputFocus** extension subroutine sets the device input focus and the last focus–change time. The X Server generates the **AIXFocusIn** and **AIXFocusOut** events.

This extension subroutine 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** value is replaced by the current X Server time.

The specified focus window must be viewable at the time the **XSetDeviceInputFocus** extension subroutine is called. Otherwise, a **BadMatch** error is generated.

Parameters

Device Specifies the input device. This parameter can have the following values:

DEVDIAL

DEVLPFK

DisplayPtr Specifies the connection to the X Server.

Focus Specifies the window for setting the input focus. This parameter can have one of the following values:

None Indicates that all dial and LPFK events are discarded until a new focus window is set. In this case, the *RevertTo* parameter is ignored.

PointerRoot Indicates that the focus window is the root window of whatever screen contains the pointer during each dial or LPFK event. In this case, the *RevertTo* parameter is ignored.

A WindowID Indicates that this window becomes the focus window of dial or LPFK. If 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.

RevertTo Specifies the window the input focus reverts to if the window becomes unviewable. If the focus window becomes unviewable later, the X Server evaluates this parameter to determine the new focus window. This parameter can have the following values:

RevertToNone Indicates that the focus reverts to that value. When the focus reverts, the X Server generates the **AIXFocusIn** and **AIXFocusOut** events, but the last focus-change time is not affected.

RevertToParent Indicates that the focus reverts to the parent window (or the closest viewable ancestor window), and the new *RevertTo* parameter value is the **RevertToNone** value.

RevertToPointerRoot Indicates that the focus reverts to that value. When the focus reverts, the X Server generates the **AIXFocusIn** and **AIXFocusOut** events, but the last focus-change time is not affected.

Time Specifies the time in a timestamp (in milliseconds) or the **CurrentTime** value.

Error Codes

AIXBadDevice

BadMatch

BadValue

BadWindow

Implementation Specifics

This extension subroutine is part of AIXwindows Runtime Environment in AIXwindows Environment/6000.

Related Information

The **XSetInputFocus** subroutine.

The **AIXFocusIn** event, **AIXFocusOut** event.

XSetDialAttributes Extension Subroutine

Purpose

Sets dial resolution.

Library

Enhanced X-Windows Library (**libXext.a**)

Syntax

XSetDialAttributes(*DisplayPtr*, *WindowID*, *DialMask*, *Resolution* , *Number*)
Display **DisplayPtr*;
Window *WindowID*;
unsigned long *DialMask*;
char **Resolution*;
int *Number*;

Description

The **XSetDialAttributes** extension subroutine sets the dial resolution for a specified window if the **Autoload** mode is set. Each bit of the *DialMask* parameter represents one possible dial. The resolution array must be as large as the largest numbered dial in the dial mask plus 1. Valid values for each byte of the array are 2 through 8 inclusive. This value ^{n} represents 2^{**n} points per revolution.

For example, if the client program wants to set the attributes for dials 0, 2, 5, and 7, it specifies the following in the *DialMask* parameter:

```
DialMask0 | DialMask2 | DialMask5 | DialMask7
```

In addition, it creates the resolution array with 8 bytes. Each of these bytes would contain a value between 2 and 8. These values can be interpreted as follows:

Value in Array	Points Per Revolution
2	4
3	8
4	16
5	32
6	64
7	128
8	256

Parameters

DialMask Specifies the dial mask, which is the bitwise-inclusive OR of one or more valid dial mask bits. Valid values for this parameter are the **DialMask0** value,and the **DialMask1** through **DialMask23** values.

DisplayPtr Specifies the connection to the X Server.

Number Specifies the number of the 1-bit in the *DialMask* parameter.

Resolution Specifies an array of dial resolutions. For each byte in the array, valid values for this parameter are **2** through **8**.

WindowID Specifies the WindowID.

Error Codes

AIXBadMode

BadValue

BadWindow

Implementation Specifics

This extension subroutine is part of AIXwindows Run Time Environment in AIXwindows Environment/6000.

XSetDialControl Extension Subroutine

Purpose

Controls the global granularity of a dial input device.

Library

Enhanced X-Windows Library (**libXext.a**)

Syntax

XSetDialControl(*DisplayPtr*, *Dialmask*, *Resolution*, *Number*)
Display **DisplayPtr*;
Bool *Dialmask*;
char **Resolution*;
int *Number*;

Description

The **XSetDialControl** extension subroutine controls the global granularity of a dial input device. This extension subroutine sets the dial granularity if the **AutoLoad** mode is off. The X Server generates the **AIXDeviceMappingNotify** event to all clients.

Parameters

DialMask	Specifies the dial mask bits.
DisplayPtr	Specifies the connection to the X Server.
Number	Specifies the number of the 1-bits in the *DialMask* parameter.
Resolution	Specifies an array of dial resolutions. For each byte in the array, valid values for this parameter are **2** through **8**.

Error Codes

AIXBadMode

BadValue

Implementation Specifics

This extension subroutine is part of AIXwindows Run Time Environment in AIXwindows Environment/6000.

Related Information

The **XSetDialAttributes** extension subroutine.

XSetLpfkAttributes Extension Subroutine

Purpose

Selects the specific lighted programmable function keys (**LPFK**) available for input and output.

Library

Enhanced X-Windows Library (**libXext.a**)

Syntax

XSetLpfkAttributes(*DisplayPtr*, *WindowID*, *LPFKMask*, *LightMask*)
Display **DisplayPtr*;
Window *WindowID*;
unsigned long *LPFKMask*;
unsigned long *LightMask*;

Description

The **XSetLpfkAttributes** extension subroutine uses the *LPFKMask* and *LightMask* parameters to select which of the keys are available for input and output when the **AutoLoad** mode is on. Each bit in the mask represents one key. To receive input from the key, the bit in the *LPFKMask* parameter should be set. To light the key, the bit in the *LightMask* parameter should be set.

Thus, when the pointer enters the window specified in the *WindowID* parameter, the keys specified by the *LightMask* parameter are turned on and the rest are turned off. The keys specified by the *LPFKMask* parameter return events, and events from the rest are ignored.

Parameters

DisplayPtr Specifies the connection to the X Server.

LightMask Specifies the **LPFK** output mask, which is the bitwise-inclusive OR of one or more valid **LPFK** mask bits. Valid values for this parameter are the **LPFKMask0** value, and the **LPFKMask2** through **LPFKMask31** values.

LPFKMask Specifies the **LPFK** input mask which is the bitwise-inclusive OR of one or more valid **LPFK** mask bits. This indicates if the X Server reports the **LPFK** keypress event to a client. Valid values for this parameter are the **LPFKMask0** value, and the **LPFKMask2** through **LPFKMask31** values.

WindowID Specifies the WindowID.

Error Codes

BadValue

BadWindow

Implementation Specifics

This extension subroutine is part of AIXwindows Run Time Environment in AIXwindows Environment/6000.

Related Information

The **XGetLpfkAttributes** extension subroutine, **XGetLpfkControl** extension subroutine.

XSetLpfkControl Extension Subroutine

Purpose

Changes the input and output of lighted programmable function keys (**LPFK**).

Library

Enhanced X-Windows Library (**libXext.a**)

Syntax

XSetLpfkControl(*DisplayPtr*, *LPFKMask*, *LightMask*)
Display **DisplayPtr*;
unsigned long *LPFKMask*;
unsigned long *LightMask*;

Description

The **XSetLpfkControl** extension subroutine changes the input and output of **LPFK**s. This extension subroutine uses the *LPFKMask* and *LightMask* parameters to select which of the keys are available for input and output when the **AutoLoad** mode is off. Each bit in the mask represents one key. To receive input from the key, the bit in the *LPFKMask* parameter should be set. To light the key, the bit in the *LightMask* parameter should be set. The server generates **AixDeviceMappingNotify** events to all clients.

Parameters

DisplayPtr Specifies the connection to the X Server.

LightMask Specifies the **LPFK** output mask. Valid values for this parameter are the **LPFKMask0** value, and the **LPFKMask2** through **LPFKMask31** values.

LPFKMask Specifies the **LPFK** input mask. Valid values for this parameter are the **LPFKMask0** value, and the **LPFKMask2** through **LPFKMask31** values.

Error Codes

AIXBadMode

BadValue

Implementation Specifics

This extension subroutine is part of AIXwindows Run Time Environment in AIXwindows Environment/6000.

Related Information

The **XGetLpfkAttributes** extension subroutine, **XSetLpfkAttributes** extension subroutine.

XSetPolyMarker Extension Subroutine

Purpose

Sets the marker in the specified graphics context.

Library

Enhanced X-Windows Library (**libXext.a**)

Syntax

XSetPolyMarker(*DisplayPtr*, *GraphicsContext*, *Marker*, *X*, *Y*)
Display **DisplayPtr*;
GC *GraphicsContext*;
Pixmap *Marker*;
int *X*, *Y*;

Description

The **XSetPolyMarker** extension subroutine 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.

Parameters

DisplayPtr	Specifies the connection to the X Server.
GraphicsContext	Specifies the graphics context.
Marker	Specifies the marker to set for the specified graphics context. Note that the depth of **Pixmap** is the value of 1.
X	Specifies the x coordinate where the x coordinate of the hotspot will be placed. This hotspot is relative to the origin of the marker and must be a point within the marker.
Y	Specifies the y coordinate where the y coordinate of the hotspot will be placed. This hotspot is relative to the origin of the marker and must be a point within the marker.

Error Codes

BadAlloc

BadGC

BadMatch

BadPixmap

Implementation Specifics

This extension subroutine is part of AIXwindows Run Time Environment in AIXwindows Environment/6000.

XStopAutoLoad Extension Subroutine

Purpose

Resets the **EventReport** mode of the dial and lighted programmable function key (LPFK) de vices.

Library

AIXwindows Library (**libXext.a**)

Syntax

XStopAutoLoad(*Display*)
Display **Display*;

Description

The **XStopAutoLoad** extension subroutine resets the **EventReport** mode of the dial and LP FK devices.

Parameter

Display Specifies the connection to the X-Server.

Implementation Specifics

This extension subroutine is part of AIXwindows Run Time Environment in AIXwindows Environment/6000.

XinitExtension Extension Subroutine

Purpose

Determines if the extension exists.

Library

Enhanced X-Windows Library (**libXext.a**)

Syntax

XExtCodes *XinitExtension(*DisplayID*, *Name*)
Display **DisplayID*;
char **Name*;

Description

The **XinitExtension** extension subroutine determines if the extension exists. Then, it allocates storage for maintaining the information about the extension on the connection, chains this onto the extension list for the connection and returns the information the stub implementor needs to access the extension.

The extension number in the **XExtCodes** data structure is needed in the other calls that follow. This extension number is unique only to a single connection.

Parameters

DisplayID	Specifies the ID of the display.
Name	Specifies the name of the extension.

Return Value

NULL	If the extension does not exist.

Implementation Specifics

This extension subroutine is part of AIXwindows Run Time Environment in AIXwindows Environment/6000.

Related Information

The **XExtCodes** data structure.

Enhanced X–Windows Events

CirculateNotify Event

Purpose

Reports when a window is restacked.

Event Format

Event, Window: **WINDOW**
Place: {**Top**, **Bottom**}

Description

The **CirculateNotify** event is reported to clients selecting the **StructureNotify** mask on the window and to clients selecting the **SubstructureNotify** mask on the parent window. This event is generated when the window is restacked as a result of a **CirculateWindow** protocol request.

Parameters

Event Specifies the window on which the event is generated.

Place Specifies the position of the window within the stack.This parameter can have the following values:

Bottom Indicates that the window is placed below all its sibling windows.

Top Indicates that the window is placed on top of all its sibling windows.

Window Specifies the window to be restacked.

Implementation Specifics

This protocol event is part of AIXwindows Run Time Environment in AIXwindows Environment/6000.

CirculateRequest Event

Purpose

Reports when a **CirculateWindow** protocol request is begun by another client on a specified window.

Event Format

Parent,Window: **WINDOW**
Place: {**Top**, **Bottom**}

Description

The **CirculateRequest** event is reported to the client selecting the **SubstructureRedirect** mask on the parent window. This event is generated when a **CirculateWindow** protocol request is issued on the *Parent* parameter, and a *Window* parameter needs to be restacked.

Parameters

Parent Specifies the parent window.

Place Specifies the new position of the window in the stack.

- **Bottom** Indicates that the window is placed below all its sibling windows.
- **Top** Indicates that the window is placed on top of all its sibling windows.

Window Specifies the window to be restacked.

Implementation Specifics

This protocol event is part of AIXwindows Run Time Environment in AIXwindows Environment/6000.

ClientMessage Event

Purpose

Reports when a client uses the **SendEvent** protocol request.

Event Format

Window: **WINDOW**
Type: **ATOM**
Format: **(8, 16, 32)**
Data: **LISTofINT8** or **LISTofINT16** or **LISTofINT32**

Description

The **ClientMessage** event is generated only when clients use the **XSendEvent** protocol request.

Parameters

Data	Represents data of twenty 8-bit values, ten 16-bit values, or five 32-bit values, although particular *Type* parameter messages may not use all of these values.
Format	Specifies whether data should be in a list of 8-bit, 16-bit, or 32-bit quantities so the server can byte-swap as necessary.
Type	Specifies an atom indicating how the data is to be interpreted by the receiving client. The server places no interpretation on the type or the data.
Window	Specifies the window to which the event was sent.

Implementation Specifics

This protocol event is part of AIXwindows Run Time Environment in AIXwindows Environment/6000.

ColormapNotify Event

Purpose

Reports colormap status.

Event Format

Window: **WINDOW**
ColorMap: **COLORMAP** or **None**
New: **BOOL**
State: {**Installed, Uninstalled**}

Description

The **ColormapNotify** event is reported to clients selecting the **ColormapChange** mask on the window. This event is generated with the value of **True** for the *New* parameter when the colormap parameter of the window is changed. It is generated with the value of **False** for the *New* parameter when the colormap of a window is installed or not installed. The *State* parameter indicates whether the colormap is currently installed.

Parameters

Colormap Specifies the colormap that is changed, installed, or not installed. This parameter can have the following values:

None Indicates that the *Window* parameter will no longer have the associated **COLORMAP** *Colormap* parameter.

If the colormap is changed, installed, or not installed using some other subroutine, the value for this parameter specifies the colormap.

New Specifies the prior status of a colormap for a specific window. This parameter can have the following values:

False Indicates that the colormap was installed or not installed.

True Indicates that the colormap was changed.

State Specifies whether the colormap is installed or not installed.

Window Specifies the window whose associated colormap is changed, installed, or not installed.

Implementation Specifics

This protocol event is part of AIXwindows Run Time Environment in AIXwindows Environment/6000.

ConfigureNotify Event

Purpose

Reports changes in the state of the window.

Event Format

Event, Window: **WINDOW**
X, Y: **INT16**
Width, Height, BorderWidth: **CARD16**
AboveSibling: **WINDOW** or **None**
OverrideRedirect: **BOOL**

Description

The **ConfigureNotify** event is reported to clients selecting the **StructureNotify** mask on the window, and to clients selecting the **SubstructureNotify** mask on the parent window. It is generated when a **ConfigureWindow** protocol request changes the state of the window.

Parameters

AboveSibling	Specifies the position of the window among the sibling windows. This parameter can have the value of **None**, which indicates that the window is at the bottom of the stack with respect to its sibling windows. Other values indicate that the window is immediately on top of the specified sibling windows.
BorderWidth	Specifies the width of the window border, in pixels.
Event	Specifies the window on which the event is generated.
Height	Specifies the inside height of the window, excluding the border.
OverrideRedirect	This parameter is from the window's attributes.
Width	Specifies the inside width of the window, excluding the border.
Window	Specifies the window that was changed.
X	Specifies the position of the x coordinate of the upper-left outer corner of the window relative to the origin of the new parent window.
Y	Specifies the position of the y coordinate of the upper-left outer corner of the window relative to the origin of the new parent window.

Implementation Specifics

This protocol event is part of AIXwindows Run Time Environment in AIXwindows Environment/6000.

ConfigureRequest Event

Purpose

Reports when a **ConfigureWindow** protocol request is begun by another client on any child window of the specified window. Reported to the client with the **SubstructureRedirect** mask selected.

Event Format

Parent, Window: **Window**
X, Y: **INT16**
Width, Height, BorderWidth: **CARD16**
Sibling: **Window** or **None**
StackMode: (**Above, Below, TopIf, BottomIf, Opposite**)
ValueMask: **BITMASK**

Description

The **ConfigureRequest** event is reported to the client selecting the **SubstructureRedirect** mask on the parent window. This event is generated when a **ConfigureWindow** protocol request is issued by another client on the child window.

Parameters

BorderWidth	Specifies the width of the border of the window in pixels. The value for this parameter is taken from the current geometry of the window.
Height	Specifies the height of the window in pixels. The value for this parameter is taken from the current geometry of the window.
Parent	Specifies the parent window.
Sibling	Has the value of **None** unless otherwise specified in the **ConfigureWindow** protocol request.
StackMode	Has the **Above** value unless otherwise specified in the **ConfigureWindow** protocol request.
ValueMask	Specifies which components were specified in the request.
X	Specifies the x coordinate of the window. The value for this parameter is taken from the current geometry of the window.
Width	Specifies the width of the window in pixels. The value for this parameter is taken from the current geometry of the window.
Window	Specifies the window to be reconfigured.
Y	Specifies the y coordinate of the window. The value for this parameter is taken from the current geometry of the window.

Implementation Specifics

This protocol event is part of AIXwindows Run Time Environment in AIXwindows Environment/6000.

Related Information

The **XConfigureRequestEvent** data structure, **XLowerWindow** subroutine, **XRestackWindows** subroutine.

CreateNotify Event

Purpose

Reports information about the creation of windows.

Event Format

Parent, *Window*: **WINDOW**
X, *Y*: **INT16**
Width, *Height*, *BorderWidth*: **CARD16**
OverrideRedirect: **BOOL**

Description

The **CreateNotify** event is reported to clients selecting the **SubstructureNotify** mask on the parent window. This event is generated whenever clients create new windows.

Parameters

BorderWidth	Specifies the width of the border of the created window, in pixels.
Height	Specifies the inside height (excluding the border) of the created window. The value for this field is always nonzero.
OverrideRedirect	Specifies whether the window overrides structure control facilities. If this parameter is the value of **True**, window manager clients normally should ignore the window.
Parent	Specifies the parent window of the created window.
Width	Specifies the inside width (excluding the border) of the created window. The value for this field is always nonzero.
Window	Specifies the ID of the created window.
X	Specifies the x coordinate of the created window relative to the origin of the parent window.
Y	Specifies the y coordinate of the created window relative to the origin of the parent window.

Implementation Specifics

This protocol event is part of AIXwindows Run Time Environment in AIXwindows Environment/6000.

DestroyNotify Event

Purpose

Reports information about which windows are deleted.

Event Format

Event, Window: **WINDOW**

Description

The **DestroyNotify** event is reported to clients selecting the **StructureNotifyMask** on the window and to clients selecting the **SubstructureNotifyMask** on the parent window. This event is generated whenever a client deletes a window.

For any given window, a **DestroyNotify** event is generated on all subhierarchies of the window before it is generated on the window itself. The ordering among sibling windows and across subhierarchies is not constrained.

Parameters

Event	Specifies the window on which the event is generated.
Window	Specifies the window to be deleted.

Implementation Specifics

This protocol event is part of AIXwindows Run Time Environment in AIXwindows Environment/6000.

EnterNotify Event

Purpose

Reports if a pointer motion or window hierarchy change causes the pointer to move from one window to another window.

Event Format

Root, Event: **WINDOW**
Child: **WINDOW** or **None**
SameScreen: **BOOL**
RootX, RootY, EventX, EventY: **INT16**
Mode: {**Normal, Grab, Ungrab**}
Detail: {**Ancestor, Virtual, Inferior, Nonlinear, NonlinearVirtual**}
Focus: **BOOL**
State: **SETofKEYBUTMASK**
Time:**TIMESTAMP**

Description

The **EnterNotify** event is generated if a pointer motion or window hierarchy change causes the pointer to move from one window to another. Only clients selecting the **EnterWindow** value on a window receive the **EnterNotify** 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 **EnterNotify** event caused by a hierarchy change is generated after the hierarchy event (the **UnmapNotify**, **MapNotify**, **ConfigureNotify**, **GravityNotify**, or **CirculateNotify** event) caused by that change. The ordering of this event with respect to the **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*, an **EnterNotify** event (with *Detail* set to the **Inferior** value) is generated on *B*.
- When the pointer moves from window *A* to window *B* and *B* is an inferior of *A*:
  - An **EnterNotify** event (with *Detail* set to the **Virtual** value) is generated on each window between *A* and *B*, exclusive (in that order).
  - An **EnterNotify** event (with *Detail* set to the **Ancestor** value) is generated on *B*.
- When the pointer moves from window *A* to window *B*, with window *C* being their least common ancestor:
  - An **EnterNotify** event (with *Detail* set to the **NonlinearVirtual** value) is generated on each window between *C* and *B*, exclusive (in that order).
  - An **EnterNotify** event (with *Detail* set to the **Nonlinear** value) is generated on *B*.
- When the pointer moves from window A to window *B* on different screens:
  - If *B* is not a root window, an **EnterNotify** event (with *Detail* set to the **NonlinearVirtual** value) is generated on each window from the root of *B* through all child windows down to but not including *B* (in that order).
  - An **EnterNotify** event (with *Detail* set to the **Nonlinear** value) is generated on *B*.

Pseudo-motion modes of the **EnterNotify** events are generated when a pointer grab activates or turns off:

- When a pointer grab activates (after any initial warp into a **ConfineTo** window, and before a **ButtonPress** event that activates the grab) with window *G* being the grab-window for the grab and window *P* being the window where the pointer is located, the **EnterNotify** events (with *Detail* set to the **NotifyGrab** value) are generated as if thepointer suddenly warped from its current position in window *P* to some position in window *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 turns off (after generating a **ButtonRelease** event that turns off the grab) with window *G* being the grab-window for the grab and window *P* being the window where the pointer is located, the **EnterNotify** events (with *Detail* set to the **NotifyUngrab** value) are generated as if the pointer suddenly warped from some position in window *G* to its current position in window *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.

Parameters

Child

Specifies the child window containing the final pointer position. This parameter can have the value of **None**, which indicates that no child window contains the final pointer position. Otherwise, the value for this parameter indicates the specified child window.

Detail

Specifies the **Notify** detail value. This parameter can have the following values:

Ancestor	**Virtual**
Inferior	**Nonlinear**
NonlinearVirtual	

Event

Specifies the window on which the event is generated.

EventX

Specifies the x coordinate of the pointer relative to the origin of the event window, if the event window is on the same screen as the root window. Otherwise, this parameter is the value of 0.

EventY

Specifies the y coordinate of the pointer relative to the origin of the event window, if the event window is on the same screen as the root window. Otherwise, this parameter is the value of 0.

Focus

Specifies whether the event window is related to the focus window. This parameter can have the following values:

True	Indicates that the event window is the focus window or a subhierarchy of the focus window.
False	Indicates that the event window is not the focus window or an inferior of the focus window.

Mode

Specifies the mode of the **EnterNotify** event. This parameter can have the following values:

Grab	Indicates a pseudo-motion event.
Normal	Indicates a normal pointer motion event.

	Ungrab	Indicates pseudo-motion events generated when a grab turns off.
Root	Specifies the root window of the screen on which the event occurred.	
RootX	Specifies the x coordinate of the pointer relative to the origin of the root window at the time of the event.	
RootY	Specifies the y coordinate of the pointer relative to the origin of the root window at the time of the event.	
SameScreen	Specifies whether the event window is on the same screen as the root window. This parameter can have the following values:	
	False	Indicates that the event and root windows are not on the same screen.
	True	Indicates that the event and root windows are on the same screen.
State	Specifies the state of the pointer buttons and modifier keys immediately prior to the event. This parameter can have values set to the bitwise- inclusive OR to one or more of the button or modifier key masks.	
Time	Specifies the time when the event was generated, in milliseconds.	

Implementation Specifics

This protocol event is part of AIXwindows Run Time Environment in AIXwindows Environment/6000.

Expose Event

Purpose

Reports information about when the contents of window regions are visible.

Event Format

Window: **WINDOW**
X, Y, *Width*, *Height*: **CARD16**
Count : **CARD16**

Description

The **Expose** event is reported to clients selecting the **Exposure** mask on the window. The circumstances under which this event is generated are not as definite as those for other events. This event can be generated when no valid contents are available for regions of a window and:

- The regions are visible.

OR

- The regions are viewable and the server is (perhaps newly) maintaining a backing store on the window.

OR

- The window is not viewable but the server is (perhaps newly) observing the **Always** or **WhenMapped** attributes of the *BackingStore* parameter of the window.

This event, however, is not generated on the **InputOnly** windows.

The window regions dissociate into a set of rectangles, and an **Expose** event is generated for each rectangle. All regions exposed by a given action are guaranteed to be reported contiguously.

All **Expose** events caused by a hierarchy change are generated after the hierarchy event (the **UnmapNotify**, **MapNotify**, **ConfigureNotify**, **GravityNotify**, or **CirculateNotify** event) caused by that change.

All **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 the **FocusOut**, **EnterNotify**, and **LeaveNotify** events is not constrained.

Parameters

Count — Specifies the number of the **Expose** events that are to follow. This parameter can have the following types of values:

Nonzero	Indicates that at least the specified number of **Expose** events (and possibly more) are to follow.
0	Indicates that no more **Expose** events follow for the given window.

Height — Specifies the height of the rectangle.

Width	Specifies the width of the rectangle.
Window	Specifies the exposed window.
X	Specifies the x coordinate of the upper-left corner of the rectangle relative to the origin of the window.
Y	Specifies the y coordinate of the upper-left corner of the rectangle relative to the origin of the window.

Implementation Specifics

This protocol event is part of AIXwindows Run Time Environment in AIXwindows Environment/6000.

FocusIn Event

Purpose

Reports input focus changes.

Event Format

Event: **WINDOW**
Mode: {**Normal, WhileGrabbed, Grab, Ungrab**}
Detail: {**Ancestor, Virtual, Inferior, Nonlinear, NonlinearVirtual, Pointer, PointerRoot, None**}

Description

The **FocusIn** events are reported to clients selecting the **FocusChange** mask on the window. These events are generated when the input focus changes.

Events with the **Normal** and **WhileGrabbed** values 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 **FocusIn** event (with the *Detail* set to the **Inferior** value) is generated on *B*.
  - If *P* is an inferior of *B*, but *P* is neither *A* nor an inferior or an ancestor of *A*, a **FocusIn** event (with the *Detail* set to the **Pointer** value) is generated on each window from *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*:
  - A **FocusIn** event (with the *Detail* set to the **Virtual** value) is generated on each window from *A* to *B*, exclusive (in order).
  - A **FocusIn** event (with the *Detail* set to the **Ancestor** value) 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*:
  - A **FocusIn** event (with the *Detail* set to the **Virtual** value) is generated on each window from *C* to *B*, exclusive.
  - A **FocusIn** event (with the *Detail* set to the **Nonlinear** value) is generated on *B*.
  - If *P* is an inferior of *B*, a **FocusIn** event (with the *Detail* set to the **Pointer** value) 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 *B* is not a root window, a **FocusIn** event (with the *Detail* set to the **NonlinearVirtual** value) is generated on each window from the root of *B* down to but not including *B* (in order).
  - A **FocusIn** event (with the *Detail* set to the **Nonlinear** value) is generated on *B*.

  - If *P* is an inferior of *B*, a **FocusIn** event (with the *Detail* set to the **Pointer** value) is generated on each window below *B* down to and including *P* (in order).

- When the focus moves from window *A* to the **NotifyPointerRoot** value (indicating that events are sent to the window under the pointer) or to the **NotifyNone** value (indicating that events are discarded) with the pointer in window *P*:
  - A **FocusIn** event (with the *Detail* set to the **PointerRoot** or **None** value) is generated on all root windows.
  - If the new focus is the **NotifyPointerRoot** value, a **FocusIn** event (with the *Detail* set to the **Pointer** value) is generated on each window from the root of *P* down to and including *P* (in order).
- When the focus moves from the **NotifyPointerRoot** or **NotifyNone** value to window *A* with the pointer in window *P*:
  - If *A* is not a root window, a **FocusIn** event (with the *Detail* set to the **NonlinearVirtual** value) is generated on each window from the root of *A* down to but not including *A* (in order).
  - A **FocusIn** event (with the *Detail* set to the **Nonlinear** value) is generated on *A*.
  - If *P* is an inferior of *A*, a **FocusIn** event (with the *Detail* set to the **Pointer** value) is generated on each window below *A* down to and including *P* (in order).
- When the focus moves from the **NotifyPointerRoot** value to the **NotifyNone** value (or vice versa) with the pointer in window *P*:
  - A **FocusIn** event (with the *Detail* set to the **PointerRoot** or **None** value) is generated on all root windows.
  - If the new focus is the **NotifyPointerRoot** value, a **FocusIn** event (with the *Detail* set to the **Pointer** value) 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* being the grab-window for the grab and *F* being current focus window, **FocusIn** events with the **Grab** value are generated as if the focus were to change from *F* to *G*.

When a keyboard grab deactivates after generating a **KeyRelease** event that deactivates the grab, with window *G* being the grab-window for the grab and window *F* being the current focus window, the **FocusIn** events (with the **Grab** value) are generated as if the focus were to change from *G* to *F*.

Parameters

Detail

Specifies the **Notify** detail value depending on the event mode. This parameter can have the following values:

Ancestor	**Virtual**
Inferior	**Nonlinear**
NonlinearVirtual	**Pointer**
PointerRoot	**None**

Event

Specifies the window on which the event is generated.

Mode

Specifies the position of the keyboard at the time the event is generated. This parameter can have the following values:

Normal	Indicates that the events are generated with the keyboard not grabbed.
Grab, Ungrab	Indicates that the events are generated after a keyboard grab activates or turns off.
WhileGrabbed	Indicates that the events are generated with the keyboard grabbed.

Implementation Specifics

This protocol event is part of AIXwindows Run Time Environment in AIXwindows Environment/6000.

FocusOut Event

Purpose

Reports input focus changes.

Event Format

Event: **WINDOW**
Mode: {**Normal, WhileGrabbed, Grab, Ungrab**}
Detail: {**Ancestor, Virtual, Inferior, Nonlinear, NonlinearVirtual, Pointer, PointerRoot, None**}

Description

The **FocusOut** events are reported to clients selecting the **FocusChange** mask on the window. These events are generated when the input focus changes.

The **FocusOut** events caused by a window unmap operation are generated after an **UnmapNotify** event. Otherwise, the order of the **FocusOut** events with respect to the **EnterNotify**, **VisibilityNotify**, and **Expose** events is not constrained.

Events with the **Normal** and **WhileGrabbed** values are generated according to the following scenarios:

- When the focus moves from window *A* to window *B* and *A* is a subhierarchy of *B* with the pointer in window *P*:
  - A **FocusOut** event (with the *Detail* parameter set to the **Ancestor** value) is generated on *A*.
  - A **FocusOut** event (with the *Detail* set to **Virtual**) is generated on each window from *A* to *B*, exclusive (in order).
- When the focus moves from window *A* to window *B* and *B* is a subhierarchy of *A* with the pointer in window *P*:
  - If *P* is a subhierarchy of *A*, but *P* is neither a subhierarchy nor an ancestor window of *B*, a **FocusOut** event (with the *Detail* parameter set to the **Pointer** value) is generated on each window from *P* up to but not including *A* (in order).
  - A **FocusOut** event (with the *Detail* parameter set to the **Inferior** value) is generated on *A*.
- 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*:
  - If *P* is a subhierarchy of *A*, a **FocusOut** event (with the *Detail* parameter set to the **Pointer** value) is generated on each window from *P* up to but not including *A* (in order).
  - A **FocusOut** event (with the *Detail* parameter set to the **Nonlinear** value) is generated on *A*.
  - A **FocusOut** event (with the *Detail* parameter set to the **NonlinearVirtual** value) is generated on each window from *A* to *C*, exclusive (in order).

- When the focus moves from window *A* to window *B* on different screens, with the pointer in window *P*:
  - If *P* is a subhierarchy of *A*, a **FocusOut** event (with the *Detail* parameter set to the **Pointer** value) is generated on each window from *P* up to but not including *A* (in order).
  - A **FocusOut** event (with the *Detail* parameter set to the **Nonlinear** value) is generated on *A*.
  - If *A* is not a root window, a **FocusOut** event (with the *Detail* parameter set to the **NonlinearVirtual** value) is generated on each window above *A* up to and including its root (in order).
- When the focus moves from window *A* to the **PointerRoot** value (indicating that events are sent to the window under the pointer) or to the **NotifyNone** value (indicating that events are discarded) with the pointer in window *P*:
  - If *P* is a subhierarchy of *A*, a **FocusOut** event (with the *Detail* parameter set to the **Pointer** value) is generated on each window from *P* up to but not including *A* (in order).
  - A **FocusOut** event (with the *Detail* parameter set to the **Nonlinear** value) is generated on *A*.
  - If *A* is not a root window, a **FocusOut** event (with the *Detail* parameter set to the **NonlinearVirtual** value) is generated on each window above *A* up to and including its root (in order).
- When the focus moves from the **NotifyPointerRoot** or **NotifyNone** value to window *A* with the pointer in window *P*:
  - If the old focus is the **NotifyPointerRoot**, a **FocusOut** event (with the *Detail* parameter set to the **Pointer** value) is generated on each window from the root of *A* down to but not including *A* (in order).
  - A **FocusOut** event (with the *Detail* parameter set to the **PointerRoot** or **None** value) is generated on all root windows.
- When the focus moves from the **PointerRoot** value to the **None** value (or vice versa) with the pointer in window *P*:
  - If the old focus is the **PointerRoot** value, a **FocusOut** event (with the *Detail* set to **Pointer**) is generated on each window from *P* up to and including its root (in order).
  - A **FocusOut** event (with the *Detail* set to **PointerRoot** or **None**) is generated on all root windows.

When a keyboard grab activates before generating a **KeyPress** event that activates the grab, with window *G* being the grab-window for the grab and window *F* being the current focus window, the **FocusOut** events with the **Grab** value are generated as if the focus were to change from window *F* to window *G*.

When a keyboard grab turns off after generating a **KeyRelease** event that turns off the grab, with window *G* being the grab-window for the grab and window *F* being the current focus window, the **FocusOut** events with the **Ungrab** value are generated as if the focus were to change from window *G* to window *F*.

Parameters

Detail

Specifies the **Notify** detail value depending on the event mode. This parameter can have the following values:

Ancestor	**Virtual**
Inferior	**Nonlinear**
NonlinearVirtual	**Pointer**
PointerRoot	**None**

Event

Specifies the window on which the event is generated.

Mode

Specifies the position of the keyboard at the time the event is generated. This parameter can have the following values:

Normal	Indicates that the events are generated with the keyboard not grabbed.
Grab, Ungrab	Indicates that the events are generated after a keyboard grab activates or turns off.
WhileGrabbed	Indicates that the events are generated with the keyboard grabbed.

Implementation Specifics

This protocol event is part of AIXwindows Run Time Environment in AIXwindows Environment/6000.

GraphicsExposure Event

Purpose

Reports when a destination region cannot be computed because the source region is obscured or out of balance.

Event Format

Drawable: **DRAWABLE**
X, *Y*, *Width*, *Height*: **CARD16**
Count: **CARD16**
MajorOpCode: **CARD8**
MinorOpCode: **CARD16**

Description

The **GraphicsExpose** event is reported to clients selecting graphics-exposures in a graphics context. This event is generated when a destination region cannot be computed due to an obscured or out-of-bounds source region. All regions exposed by a given graphics subroutine are guaranteed to be reported contiguously.

Parameters

Count Specifies the number of **GraphicsExpose** events to follow. This parameter can have the following types of values:

- **Nonzero** Indicates that at least the specified number of **GraphicsExpose** events (and possibly more) are to follow.
- **0** Indicates that no more **GraphicsExpose** events are to follow.

Drawable Specifies the drawable of the destination region on which the graphics subroutine is to be performed.

Height Specifies the extent of the rectangle.

MajorOpCode Specifies which graphics subroutine was initiated by the client. This field can have the following values for the **Core** protocol request:

- **CopyArea** Indicates that the client began the **CopyArea** protocol request.
- **CopyPlane** Indicates that the client began the **CopyPlane** protocol request.

MinorOpCode Specifies which graphics subroutine was began by the client. This parameter, however, is not defined by the **Core** protocol request, and its value in these cases is 0, although it may be used by extension subroutines.

Width Specifies the extent of the rectangle.

X Specifies the x coordinate of the upper left corner of the rectangle relative to the origin of the *Drawable*.

Y Specifies the x coordinate of the upper left corner of the rectangle relative to the origin of the *Drawable*.

Implementation Specifics

This protocol event is part of AIXwindows Run Time Environment in AIXwindows Environment/6000.

GravityNotify Event

Purpose

Reports when a window is moved as a result of resizing the parent window.

Event Format

Event, Window: **WINDOW**
X, Y: **INT16**

Description

The **GravityNotify** event is reported to clients selecting the **SubstructureNotify** mask on the parent and to clients selecting the **StructureNotify** mask on the window. This event is generated when a window is moved as a result of resizing the parent window.

Parameters

Event	Specifies the window on which the event is generated.
Window	Specifies the window that was moved.
X	Specifies the x coordinate of the upper-left outer corner of the window relative to the origin of the new parent window.
Y	Specifies the y coordinate of the upper-left outer corner of the window relative to the origin of the new parent window.

Implementation Specifics

This protocol event is part of AIXwindows Run Time Environment in AIXwindows Environment/6000.

KeyPress, KeyRelease, ButtonPress, ButtonRelease, or MotionNotify Events

Purpose

Reports when a key or button changes state or when the pointer moves.

Event Format

Root, *Event*: **WINDOW**
Child: **WINDOW** or **None**
SameScreen: **BOOL**
RootX, *RootY*, *EventX*, *EventY*: **INT16**
Detail: <see below>
State: **SETofKEYBUTMASK**
Time: **TIMESTAMP**

Description

The **KeyPress** and **KeyRelease** events are generated when a key changes state. The **KeyPress** and **KeyRelease** events are generated for all keys, even those mapped to modifier bits. These events are reported to clients selecting the **KeyPress** or **KeyRelease** mask on the window.

The **ButtonPress** and **ButtonRelease** events are generated when a button changes state. These events are reported to clients selecting the **ButtonPress** or **ButtonRelease** mask on the window.

The **MotionNotify** events are generated when the pointer moves and the pointer motion begins and ends in the window. The granularity of the **MotionNotify** events is not guaranteed, but a client selecting this event type is guaranteed to receive at least one event when the pointer moves and then rests. To receive the **MotionNotify** events, one or more of the following masks must be selected on the window:

- The **Button (1-5) Motion** mask, which indicates that the **MotionNotify** event is reported only when one or more of the specified buttons is pressed.
- The **ButtonMotion** mask, which indicates that the **MotionNotify** event is reported only when at least one button is pressed.
- The **PointerMotion** mask, which indicates that the **MotionNotify** event is reported independent of the state of the pointer buttons.
- The **PointerMotionHint** mask, which indicates that only one **MotionNotify** event (with the **Hint** detail value) can be reported for the event window until the key or button state changes, or the pointer leaves the event window, or the client initiates the **QueryPointer** or **GetMotionEvents** protocol request.

For all these events, the source of the event is the window in which the pointer is located. The server finds the window to which the event normally is reported (the event window) by searching the hierarchy (beginning with the source window). The event window is the firstwindow on which any client has selected interest in the event, if no other window prohibitsevent generation by including the event type in its **DoNotPropagate** mask. The actual window used for reporting can be modified by active grabs, and, in the case of keyboard events, by the focus window.

Parameters

Child
Specifies whether the source window is a subhierarchy of the event window. This parameter can have the value of **None**, which indicates that the source window is not a subhierarchy of the event window. Otherwise, the value for the *Child* parameter is set to the child of the event window that is the source window or its ancestor.

Detail
Specifies the detail corresponding to the event selected. This parametercan have one of the following names, depending on the event type selected:

Button
Specifies the pointer button that changed state for the **ButtonPress** and **ButtonRelease** events. This parameter can have the **Button1**, the **Button2**, the **Button3**, the **Button4**, or the **Button5** value.

Hint
Specifies the detail for the **MotionNotify** event. This parameter can have the **NotifyNormal** or the **NotifyHint** value.

Keycode
Specifies the detail for the **KeyPress** or the **KeyRelease** events. This parameter is set to a number that represents a physical key on the keyboard.

Event
Specifies the window on which the event is generated.

EventX
Specifies the x coordinate of the pointer relative to the origin of the event window if the event window is on the same screen as the root window. Otherwise, this parameter is the value of 0.

EventY
Specifies the y coordinate of the pointer relative to the origin of the event window if the event window is on the same screen as the root window. Otherwise, this parameter is the value of 0.

Root
Specifies the root window of the source window.

RootX
Specifies the x coordinate of the pointer relative to the origin of the root window at the time of the event.

RootY
Specifies the y coordinate of the pointer relative to the origin of the root window at the time of the event.

SameScreen
Specifies whether the event window is on the same screen as the root window. This parameter can have the following values:

False
Indicates that the event and root windows are not on the same screen.

True
Indicates that the event and root windows are on the same screen.

State
Specifies the state of the pointer buttons and modifier keys immediately prior to the event. This parameter has the value of the

	bitwise-inclusive OR of one or more of the button or modifier key masks.
Time	Specifies the time the event is generated, in miiliseconds.

Implementation Specifics

This protocol event is part of AIXwindows Run Time Environment in AIXwindows Environment/6000.

KeymapNotify Event

Purpose

Reports information about changes in the keyboard state.

Event Format

Keys: **LISTofCARD8**

Description

The **KeymapNotify** event is reported to clients selecting the **KeymapStateMask** on a window. This event is generated immediately after every **EnterNotify** and **FocusIn** events.

Parameter

Keys — Specifies the bit vector of the keyboard. Each bit set to the value of 1 indicates that the corresponding key is currently pressed. The vector is represented as 32 bytes. Byte *N* (from the value of 0) contains the bits for keys 8*N* to 8*N* + 7 with the least significant bit in the byte representing key 8*N*.

Implementation Specifics

This protocol event is part of AIXwindows Run Time Environment in AIXwindows Environment/6000.

LeaveNotify Event

Purpose

Reports if a pointer motion or window hierarchy change causes the pointer to move from one window to another window.

Event Format

Root, Event: **WINDOW**
Child: **WINDOW** or **None**
SameScreen: **BOOL**
RootX, *RootY*, *EventX*, *EventY*: **INT16**
Mode: {**Normal, Grab, Ungrab**}
Detail: {**Ancestor, Virtual, Inferior, Nonlinear, NonlinearVirtual**}
Focus: **BOOL**
State: **SETofKEYBUTMASK**
Time:**TIMESTAMP**

Description

The **LeaveNotify** event is generated if a pointer motion or window hierarchy change causes the pointer to move from one window to another. Only clients selecting the **LeaveWindow** mask on a window receive the **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 **LeaveNotify** event caused by a hierarchy change is generated after the hierarchy event (the **UnmapNotify**, **MapNotify**, **ConfigureNotify**, **GravityNotify**, **CirculateNotify** hierarchy event) caused by that change. The ordering of this event with respect to the **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 the *Detail* parameter of the **Ancestor** value) is generated on *A*.
  - A **LeaveNotify** event (with the *Detail* parameter of the **Virtual** value) is generated on each window between *A* and *B*, exclusive (in that order).
- When the pointer moves from window *A* to window *B* and *B* is an inferior of *A*:
  - A **LeaveNotify** event (with the *Detail* parameter of the **Inferior** value) is generated on *A*.
- When the pointer moves from window *A* to window *B* with window *C* being their least common ancestor:
  - A **LeaveNotify** event (with the *Detail* parameter of the **Nonlinear** value) is generated on *A*.
  - A **LeaveNotify** event (with the *Detail* parameter of the **NonlinearVirtual** value) is generated on each window between *A* and *C*, exclusive (in that order).

- When the pointer moves from window A to window *B* on different screens:
  - If *A* is not a root window, a **LeaveNotify** event (with the *Detail* parameter of the **NonlinearVirtual** value) is generated on each window above *A* up to and including its root (in that order).
  - A **LeaveNotify** event (with the *Detail* parameter of the **Nonlinear** value) is generated on *A*.

Pseudo-motion modes of the **LeaveNotify** events are generated when a pointer grab activates or turns off:

- When a pointer grab activates (after any initial warp into a **ConfineTo** window and before a **ButtonPress** event that activates the grab) with window *G* being the grab-window for the grab and window *P* being the window where the pointer is located, the **LeaveNotify** events (with the *Mode* **Grab**) are generated as if the pointer suddenly warped from its current position in window *P* to some position in window *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 turns off (after generating a **ButtonRelease** event that turns off the grab) with window *G* being the grab-window for the grab and window *P* being the window where the pointer is located, the **LeaveNotify** events (with the *Mode* parameterof the **Ungrab** value) are generated as if the pointer suddenly warped from some position in window *G* to its current position in window *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.

Parameters

Child
Specifies the child window containing the final pointer position. This parameter can have the value of **None**, which indicates that no child window contains the final pointer position. Otherwise, the value for this parameter indicates the specified child window.

Detail
Specifies the **Notify** detail value. This parameter can have the following values:

Ancestor	**Virtual**
Inferior	**Nonlinear**
NotifyNonlinear	

Event
Specifies the window on which the event is generated.

EventX
Specifies the x coordinate of the pointer relative to the origin of the event window, if the event window is on the same screen as the root window. Otherwise, this parameter is the value of 0.

EventY
Specifies the y coordinate of the pointer relative to the origin of the event window, if the event window is on the same screen as the root window. Otherwise, this parameter is the value of 0.

Focus Specifies whether the event window is related to the focus window. This parameter can have the following values:

True	Indicates that the event window is the focus window or a subhierarchy of the focus window.
False	Indicates that the event window is not the focus window or a subhierarchy of the focus window.

Mode Specifies the mode of the **LeaveNotify** event. This parameter can have the following values:

Grab	Indicates a pseudo-motion event.
Normal	Indicates a normal pointer motion event.
Ungrab	Indicates pseudo-motion events generated when a grab turns off.

Root Specifies the root window of the screen on which the event occurred.

RootX Specifies the x coordinate of the pointer relative to the origin of the root window at the time of the event.

RootY Specifies the y coordinate of the pointer relative to the origin of the root window at the time of the event.

SameScreen Specifies whether the event window is on the same screen as the root window. This parameter can have the following values:

False	Indicates that the event and root windows are not on the same screen.
True	Indicates that the event and root windows are on the same screen.

State Specifies the state of the pointer buttons and modifier keys immediately prior to the event. This parameter can have values set to the bitwise-inclusive OR or to one or more of the button or modifier key masks.

Time Specifies the time when the event was generated, in milliseconds.

Implementation Specifics

This protocol event is part of AIXwindows Run Time Environment in AIXwindows Environment/6000.

MapNotify Event

Purpose

Reports information about which windows are mapped.

Event Format

Event, Window: **WINDOW**
OverrideRedirect: **BOOL**

Description

The **MapNotify** event is reported to clients selecting the **StructureNotifyMask** on the window and to clients selecting the **SubstructureNotifyMask** on the parent window. This event is generated when the window changes from an unmapped state to a mapped state.

Parameters

Event	Specifies the window on which the event is generated.
OverrideRedirect	Specifies whether the window overrides structure control facilities. Clients normally should ignore the window if this parameter is the value of **True**.
Window	Specifies the window that was mapped.

Implementation Specifics

This protocol event is part of AIXwindows Run Time Environment in AIXwindows Environment/6000.

MapRequest Event

Purpose

Reports when **MapWindow** protocol requests are called by other clients.

Event Format

Parent, Window: **WINDOW**

Description

The **MapRequest** event is reported to the client selecting the **SubstructureRedirectMask** on the parent window. This event is generated when a **MapWindow** protocol request is issued on an unmapped window with the *OverrideRedirect* parameter set to thevalue of **False**.

Parameters

Parent	Specifies the parent window.
Window	Specifies the window to be mapped.

Implementation Specifics

This protocol event is part of AIXwindows Run Time Environment in AIXwindows Environment/6000.

MappingNotify Event

Purpose

Reports mapping changes.

Event Format

Request: {**Modifier**, **Keyboard**, **Pointer**}
FirstKeycode, *Count*: **CARD8**

Description

The **MappingNotify** event is sent to all clients. No mechanism exists to disengage this event.

Parameters

Count — Specifies the number of keyboards altered. This parameter is set only if the value for the *Request* parameter is the **Keyboard** value.

FirstKeycode — Indicates the range of the altered keycode. This parameter is set only if the value for the *Request* parameter is the **Keyboard** value.

Request — Specifies the kind of mapping change that occurred. This parameter can have the following values:

Keyboard — Indicates that the **ChangeKeyboardMapping** protocol request was successful.

Modifier — Indicates that the **SetModifierMapping** protocol request was successful.

Pointer — Indicates that the **SetPointerMapping** protocol request was successful.

Implementation Specifics

This protocol event is part of AIXwindows Run Time Environment in AIXwindows Environment/6000.

Related Information

The **SetModifierMapping** protocol request, **ChangeKeyboardMapping** protocol request, **SetPointerMapping** protocol request.

NoExposure Event

Purpose

Reports whenever no **GraphicsExpose** event is produced, where one may have been produced by a graphics subroutine.

Event Format

Drawable: **DRAWABLE**
MajorOpCode: **CARD8**
MinorOpCode: **CARD16**

Description

The **NoExposure** event is generated whenever no **GraphicsExpose** event is produced, where one may have been produced by a graphics subroutine. In other words, the client is really asking for a **GraphicsExpose** event but instead receives a **NoExpose** event.

Parameters

Drawable Specifies the drawable of the destination region on which the graphics subroutine is to be performed.

MajorOpCode Specifies which graphics subroutine was initiated by the client. This parameter can have the following values for the core protocol request:

CopyArea Indicates that the client initiated the **CopyArea** protocol request.

CopyPlane Indicates that the client initiated the **CopyPlane** protocol request.

MinorOpCode Specifies which graphics subroutine was initiated by the client. This parameter, however, is not defined by the **Core** protocol request, and in these cases the value is 0.

Implementation Specifics

This protocol event is part of AIXwindows Run Time Environment in AIXwindows Environment/6000.

PropertyNotify Event

Purpose

Reports information about property changes for a specified window.

Event Format

Window: **WINDOW**
Atom: **ATOM**
State: {**NewValue**, **Delete**}
Time: **TIMESTAMP**

Description

The **PropertyNotify** event is reported to clients selecting the **PropertyChange** mask on the window. This event is generated when a property of the window is changed by the **ChangeProperty**, **DeleteProperty**, **GetProperty**, or **RotateProperties** protocol request.

Parameters

Atom
Specifies the atom of the property and indicates which property was changed or selected.

State
Specifies whether the property was changed to a new value or was deleted. This parameter can have the following values:

Deleted
Indicates that a property of the window was deleted using the **DeleteProperty** protocol request or, if the *Delete* parameter is the value of **True**, by using the **GetProperty** protocol request.

NewValue
Indicates that a property of the window was changed (or all or part of a property was replaced with identical data) using the **ChangeProperty** or **RotateProperties** protocol request even when adding zero-length data or when replacing all or part of a property with identical data.

Time
Specifies the server time when the property was changed.

Window
Specifies the window for which the associated property was changed.

Implementation Specifics

This protocol event is part of AIXwindows Run Time Environment in AIXwindows Environment/6000.

ReparentNotify Event

Purpose

Reports when a window is re-parented.

Event Format

Event, *Window*, *Parent*: **WINDOW**
X, *Y*: **INT16**
OverrideRedirect: **BOOL**

Description

The **ReparentNotify** event is reported to clients selecting the **SubstructureNotify** mask on the old or the new parent window and to clients selecting the **StructureNotify** mask onthe window. This event is generated when the window is re-parented.

Parameters

Event	Specifies the window on which the event is generated.
OverrideRedirect	Is the flag specified in the window attributes.
Parent	Specifies the new parent window.
Window	Specifies the window that was re-parented.
X	Specifies the x coordinate of the upper left corner of the re-parented window relative to the origin of the new parent window.
Y	Specifies the y coordinate of the upper left corner of the re-parented window relative to the origin of the new parent window.

Implementation Specifics

This protocol event is part of AIXwindows Run Time Environment in AIXwindows Environment/6000.

ResizeRequest Event

Purpose

Reports when another client attempts to change the size of a window.

Event Format

Window: **WINDOW**
Width, *Height*: **CARD16**

Description

The **ResizeRequest** event is reported to the client selecting the **ResizeRedirect** mask on the window. This event is generated whenever another client attempts to change the size of the specified window by using the **ConfigureWindow** protocol request.

Parameters

Height	Specifies the inside height (excluding the border) of the window.
Width	Specifies the inside width (excluding the border) of the window.
Window	Specifies the window to be resized.

Implementation Specifics

This protocol event is part of AIXwindows Run Time Environment in AIXwindows Environment/6000.

SelectionClear Event

Purpose

Reports when a new owner is being defined by the **SetSelectionOwner** protocol request.

Event Format

Owner: **Window**
Selection: **ATOM**
Time: **TIMESTAMP**

Description

The **SelectionClear** event is reported to the current owner of a selection. This event is generated whenever a client initiates the **SetSelectionOwner** protocol request. This event is generated on the window losing ownership of the selection to a new owner.

Parameters

Owner	Specifies the window that is specified by the current owner in its initiation of the **SetSelectionOwner** protocol request.
Selection	Specifies the selection atom.
Time	Specifies the last change time recorded for the selection.

Implementation Specifics

This protocol event is part of AIXwindows Run Time Environment in AIXwindows Environment/6000.

SelectionNotify Event

Purpose

Reports in response to a **ConvertSelection** protocol request when there is no owner for the selection, or to the **SendEvent** protocol request when there is an owner.

Event Format

Requestor: **WINDOW**
Selection, *Target*: **ATOM**
Property: **ATOM** or **None**
Time: **TIMESTAMP** or **CurrentTime**

Description

The **SelectionNotify** event is generated by the server in response to a **ConvertSelection** protocol request when there is no owner for the selection. If there is an owner, this event should be generated with the **SendEvent** protocol request.

The owner of a selection should send this event to a requestor when a selection is converted and stored as a property, or when a selection conversion cannot be performed, indicated by the *Property* parameter of the value of **None**.

Parameters

Property	Specifies the atom that indicates which property the result was stored on. If the conversion could not be performed, this parameter is set to the value of **None**.
Requestor	Specifies the window associated with the requestor of the selection.
Selection	Specifies the atom that indicates the selection.
Target	Specifies the atom that indicates the converted type.
Time	Specifies the time the conversion took place.

Implementation Specifics

This protocol event is part of AIXwindows Run Time Environment in AIXwindows Environment/6000.

SelectionRequest Event

Purpose

Reports whenever a client requests a selection conversion by initiating the **ConvertSelection** protocol request and the specified selection is owned by a window.

Event Format

Owner: **WINDOW**
Selection: **ATOM**
Target: **ATOM**
Property: **ATOM** or **None**
Requestor: **WINDOW**
Time: **TIMESTAMP** or **CurrentTime**

Description

The **SelectionRequest** event is reported to the owner of a selection. This event is generated whenever a client issues a **ConvertSelection** protocol request.

The client who owns the selection should do the following:

- Convert the selection based on the atom specified in the *Target* parameter.
- If a property is specified, store the result as that property on the requestor window and send a **SelectionNotify** event to the requestor using the **SendEvent** protocol request with an empty event mask. (That is, the event should be sent to the creator of the requestor window.)
- If no property is specified (that is, the *Property* parameter is the value of **None**), choose a property name, store the results as that property on the requestor window and send a**SelectionNotify** event giving the actual name.
- If the selection cannot be converted as requested, send a **SelectionNotify** event with the *Property* parameter set to the value of **None**.

Parameters

Owner	Specifies the window owning the selection. This parameter is set to the window that is specified by the current owner in its initiation of the **SetSelectionOwner** protocol request.
Property	Specifies the property associated with the window.
Requestor	Specifies the window requesting the selection.
Selection	Specifies the atom that names the selection.
Target	Specifies the atom that indicates the specified type of the selection.
Time	Specifies the time the conversion took place.

Implementation Specifics

This protocol event is part of AIXwindows Run Time Environment in AIXwindows Environment/6000.

UnmapNotify Event

Purpose

Reports when a window changes from a mapped state to an unmapped state.

Event Format

Event, Window: **WINDOW**
FromConfigure: **BOOL**

Description

The **UnmapNotify** event is reported to clients selecting the **StructureNotify** mask on the window and to clients selecting the **SubstructureNotify** mask on the parent window. This event is generated when the window changes from a mapped state to an unmapped state.

Parameters

Event	Specifies the window on which the event is generated.
FromConfigure	Set to the value of **True** if the event was generated as a result of resizing the parent of a window when the window itself had a *WindowGravity* parameter of the **Unmap** value.
Window	Specifies the window that is unmapped.

Implementation Specifics

This protocol event is part of AIXwindows Run Time Environment in AIXwindows Environment/6000.

VisibilityNotify Event

Purpose

Reports whenever there is a change in the visibility of a specified window.

Event Format

Window: **WINDOW**
State: {**Unobscured**, **PartiallyObscured**, **FullyObscured**}

Description

The **VisibilityNotify** event is reported to clients selecting the **VisibilityChange** mask on the window. This event is generated whenever the visibility of a specified window changesstate. However, it is never generated on the **InputOnly** windows.

The **VisibilityNotify** events caused by a hierarchy change are generated after the hierarchy event (the **UnmapNotify**, **MapNotify**, **ConfigureNotify**, **GravityNotify**, or **CirculateNotify** hierarchy event) that caused the change.

The **VisibilityNotify** events on a window are generated before the **Expose** events on thatwindow, but not all **VisibilityNotify** events on all windows must be generated before all **Expose** events on all windows. The order of the **VisibilityNotify** events with respect to the **FocusOut**, **EnterNotify**, and **LeaveNotify** events is not constrained.

Parameters

State Specifies the visibility state of the window. All subwindows of a window are ignored when determining the visibility state of the window. This parameter can have the following values:

FullyObscured	Indicates that the window changed state from viewable and completely unobscured, viewable and partially obscured, or not viewable to viewable and fully obscured.
PartiallyObscured	Indicates that the window changed state from viewable and completely unobscured or not viewable to viewable and partially obscured.
Unobscured	Indicates that the window changed state from partially obscured, fully obscured, or notviewable to viewable and completely unobscured.

Window Specifies the window whose visibility state changes.

Implementation Specifics

This protocol event is part of AIXwindows Run Time Environment in AIXwindows Environment/6000.

Curses Subroutine Library

Curses Subroutine Library

Purpose

Controls cursor movement and windowing.

Library

Curses Library (**libcurses.a**)

Syntax

#include <curses.h>
#include <term.h>

Description

The **curses** subroutine library allows you to manipulate data structures called *windows*, which can also be thought of as two–dimensional arrays of characters representing all or part of a screen. A default window called *stdscr* is supplied, and other windows can be created using the **curses** library **newwin** routine. Routine names which begin with the letter **w** refer to *windows*. For example, **wsetscrreg**. Routine names which begin with **m_** refer to macros. The macros perform the same function as the routines. For example, **m_move**.

minicurses is a sub–set of the full **curses** subroutine library. It allows you to perform screen optimization programs, but it does not allow you to issue windowing or input functions. **minicurses** routines are marked with an asterisk (*).

For ease of use, the **curses** routines have been separated by function:

- Initialization Routines
- Option Setting Routines
- Terminal Mode Setting Routines
- Window Manipulation Routines
- Displaying Output to the Terminal Routines
- Writing on Window Structures Routines
  - Moving the Cursor
  - Writing One Character
  - Writing a String
  - Clearing Areas of the Screen
  - Inserting and Deleting Text
  - Formatted Output
  - Input from a Window
  - Input from the Terminal
- Video Attributes Routines
  - Bells and Flashing Lights
- Portability Functions Routines
- Cursor Movement Routine
- Miscellaneous Functions Routines
- Terminfo Level Routines
- Termcap Compatibility Routines

Initialization Routines

The following functions are called when initializing a program.

initscr ( )*

Determines the terminal type and initializes **curses** data structures. Also arranges that the first call to **refresh** will clear the screen.

endwin ( )*

Restores tty modes, moves the cursor to the lower left corner of the screen, resets the terminal into the proper non–visual mode, and tears down all appropriate data structures. A program should always call **endwin** before exiting.

newterm(*type*, *outfd*, *infd*)

Set up new terminal of given type to output on *outfd* and input from *infd*. If output is to be directed to more than one terminal, **newterm** should be called instead of **initscr**. **newterm** should be called once for each terminal. It returns a variable of type SCREEN * which should be saved as a reference to that terminal. The arguments are the type of the terminal (a string) and a stdio file descriptor (FILE *) for output to the terminal. The program should also call **endwin** for each terminal being used.

set_term(*new*)

This function is used to switch to a different terminal. The screen reference *new* becomes the new current terminal. The previous terminal is returned by the function. All other calls affect only the current terminal.

longname( )

This function returns a pointer to a static area containing a verbose description of the current terminal. It is defined only after a call to **initscr**, **newterm**, or **setupterm**.

Option Setting Routines

The following functions set options within **curses**. In each case, *win* is the window affected, and *bf* is the boolean flag with the value TRUE or FALSE indicating whether to enable or to disable the option. All options are initially FALSE. It is not necessary to turn these options off before calling **endwin**.

clearok(*win, bf*)

If set, the new call to **wrefresh** with this window will clear the screen and redraw the entire screen. If *win* is *curscr*, the next call to **wrefresh** with any window will cause the screen to be cleared. This is useful when the contents of the screen are uncertain, or in some cases for a more pleasing visual effect.

idlok(*win*, *bf*)*

If enabled, **curses** will consider using the hardware insert/delete line feature of terminals so equipped. If disabled, **curses** will never use this feature. The insert/delete character feature is always considered. Enable this option only if your application needs insert/delete line, for example, for a screen editor. If insert/delete line cannot be used, **curses** will redraw the changed portions of all lines that do not match the desired line.

keypad(*win*, *bf*)

This option enables the keypad of the user's terminal. If enabled, the user can press a function key (such as an arrow key) and **getch** will return a single value representing the function key. If disabled, **curses** will not treat function keys specially. If the keypad in the terminal can be turned on (made to transmit) and off (made to work locally), turning on this option will turn on the terminal keypad.

leaveok(*win*, *flag*)

Normally, the hardware cursor is left at the location of the window cursor being refreshed. This option allows the cursor to be left wherever the update happens to leave it. It is useful for applications where the cursor is not used, since it reduces the need for cursor motions. If possible, the cursor is made invisible when this option is enabled.

meta(*win*, *flag*)*

If enabled, characters returned by **getch** are transmitted with all 8 bits, instead of stripping the highest bit. The value of OK is returned if the request succeeded, the value ERR is returned if the terminal or system is not capable of 8–bit input.

nodelay(*win, bf*)

This option causes **getch** to be a non–blocking call. If no input is ready, **getch** will return –1. If disabled, **getch** will hang until a key is pressed.

intrflush(*win*, *bf*)

If this option is enabled when an interrupt key is pressed on the keyboard (interrupt, quit, suspend), all output in the *tty* driver queue will be flushed, giving the effect of faster response to the interrupt but causing **curses** to have the wrong idea of what is on the screen. DIsabling the option prevents the flush. The default is for the option to be enabled. This option depends on support in the underlying teletype driver.

typeahead(*fd*)

Sets the file descriptor for typeahead check. *fd* should be an integer returned from *open* or *fileno*. Setting typeahead to –1 will disable typeahead check. By default, file descriptor 0 (*stdin*) is used. Typeahead is checked independently for each screen, and for multiple interactive terminals it should probably be set to the appropriate input for each screen. A call to **typeahead** always affects only the current screen.

scrollok(*win*, *flag*)

This option controls what happens when the cursor of a window is moved off the edge of the window, either from a newline on the bottom line, or typing the last character of the last line. If disabled, the cursor is left on the bottom line. If enabled, **wrefresh** is called on the window, and then the physical terminal and window are scrolled up one line. Note that in order to get the physical scrolling effect on the terminal, it is also necessary to call **idlok**.

setscrreg(*t*, *b*)

wsetscrreg(*win*, *t*, *b*)

These functions allow the user to set a software scrolling region in a window *win* or *stdscr*. *t* and *b* are the line numbers of the top and bottom margin of the scrolling region. (Line 0 is the top line of the window.) If this option and **scrollok** are enabled, an attempt to move off the bottom margin line will cause all lines in the scrolling region to scroll up one line.

Terminal Mode Setting Routines

These functions are used to set modes in the *tty* driver. The initial mode usually depends on the setting when the program was called: the initial modes documented here represent the normal situation.

cbreak ( )*

nocbreak ( )*

These two functions put the terminal into and out of CBREAK mode. In this mode, characters typed by the user are immediately available to the program. When out of this mode, the teletype driver will buffer characters typed until newline is typed. Interrupt and flow control characters are unaffected by this mode. Initially the terminal is not in CBREAK mode. Most interactive programs using **curses** will set this mode.

echo ( )*

noecho ( )*

These functions control whether characters typed by the user are echoed as typed. Initially, characters typed are echoed by the teletype driver.

nl ( )*

nonl ( )*

These functions control whether newline is translated into carriage return and linefeed on output, and whether return is translated into newline on input. Initially, the translations do occur. By disabling these translations, **curses** is able to make better use of the linefeed capability, resulting in faster cursor motion.

raw ( )*

noraw ( )*

The terminal is placed into or out of raw mode. Raw mode is similar to **cbreak** mode in that characters that are typed are immediately passed to the user program. In RAW mode the interrupt, quit, and suspend characters are passed uninterpreted instead of generating a signal. RAW mode also causes 8–bit input and output.

resetty ( )*

savetty ( )*

These functions save and restore the state of the *tty* modes. **savetty** saves the current state in a buffer, **resetty** restores the *tty* state to what it was prior to calling **savetty**.

Window Manipulation Routines

The following routines allow you to create, change, move, and otherwise manipulate windows.

newwin(*lines*, *cols*, *begin_y*, *begin_x***)**

Creates a new window with the given number of lines and columns. The upper left corner of the window is at line *begin_y* column *begin_x*. If the value for either *lines* or *cols* is zero, they will default to the value of Lines–*begin_y* and COLS–*begin_x*. If you create a new window with all values set to zero, a full–screen window will be created.

newpad(*numlines*, *numcols*)

Creates a new *pad* data structure. A pad is similar to a window, except that it is not restricted by the screen size, and is not associated with a particular part of the screen. Pads can be used when a large window is needed, and only a part of the window will be on the screen at one time. Automatic refreshes of pads (i.e., from scrolling or echoing of input) do not occur. It is not valid to call **refresh** with a pad as an argument, instead, the **prefresh** or **pnoutrefresh** routines should be called. Note that these routines require additional parameters to specify the part of the pad to be displayed and the location on the screen to be used for display.

subwin(*win*, *lines*, *cols*, *begin_y*, *begin_x*)

Creates a new window within a window. The new window is at position *begin_y*, *begin_x* on the screen. The window is relative to the screen, and is created in the middle of window *win*. Any changes made to one window will affect both windows. When using this function it is often necessary to call **touchwin** before calling **wrefresh**.

delwin(*win*)

Deletes the named window, freeing up all memory associated with it. If there are overlapping windows, subwindows should be deleted first.

mvwin(*win*, *by*, *bx*)

Moves the window so that the upper left corner will be at position (*by*, *bx*). If a move would cause the window to be moved off the screen, an error occurs, and the window is not moved.

touchwin(*win*)

This routine discards all optimization information about which parts of the window have been touched, by pretending that the entire window has been drawn on. This is sometimes necessary when using overlapping windows, since a change to one window will affect the other window, but the records of which lines have been changed in the other window will not reflect the change.

overlay(*win1*, *win2*)

overwrite(*win1*, *win2*)

These functions overlay *win1* on top of *win2*. All text in *win1* is copied into *win2*. Blanks (spaces) are not copied when using the **overlay** function. Blanks (spaces) are copied when using the **overwrite** function.

Displaying Output to the Terminal Routines

The following routines cause output to be directed to the terminal.

refresh( )*

m_refresh( )*

wrefresh(*win***)**

These functions must be called to get any output on the terminal, because other routines merely manipulate data structures. **wrefresh** makes the current screen look like *window*. Optimization is performed according to what is already present in the window. **refresh** performs the same function, except it makes the current screen look like *stdscr*. The physical terminal cursor is left at the location of the window's cursor, unless **leaveok** has been enabled.

doupdate( )

wnoutrefresh(*win***)**

These two functions allow multiple updates with more efficiency than **wrefresh**. In addition to all window structures, **curses** keeps two data structures representing the terminal screen: a *physical* screen, describing what is actually on the screen, and a *virtual* screen, describing what the programmer wants to have on the screen. **wrefresh** works by first copying the named window to the virtual screen (**wnoutrefresh**), and then calling the routine to update the screen (**doupdate**). If the request is to have several windows output at the same time, a series of calls to **wrefresh** will result in alternating calls to **wnoutrefresh** and **doupdate**. By calling **wnoutrefresh** for each window, it is then possible to call **doupdate** once, resulting in only one burst of output.

prefresh(*pad, pminrow, pmincol, sminrow, smincol, smaxrow, smaxcol***)**

pnoutrefresh(*pad, pminrow, pmincol, sminrow, smincol, smaxrow, smaxcol***)**

These routines are the same as **wrefresh** and **wnoutrefresh**, except that pads, instead of windows, are involved. The additional parameters are necessary to indicate what part of the pad and screen are involved. *pminrow* and *pmincol* specify the upper left corner, in the pad, of the rectangle to be displayed. *sminrow*, *smincol*, *smaxrow*, and *smaxcol* specify the edges, on the screen, of the rectangle to be displayed in. The lower right corner in the pad of the rectangle to be displayed is calculated from the screen coordinates, since the rectangles must be the same size. Both rectangles must be entirely contained within their respective structures.

Writing on Window Structures Routines

These routines are used to "draw" text on windows. In all cases, a missing *win* is taken to be *stdscr*. *y* and *x* are the row and column, respectively. The upper left corner is always (0,0), not (1,1). The **mv** functions imply a call to **move** before the call to the other function.

Moving the Cursor

move(*y, x*)*

m_move(*y, x*)*

wmove(*win, y, x*)

The cursor associated with the window is moved to the given location. This does not move the physical cursor of the terminal until **refresh** is called. The position specified is relative to the upper left corner of the window,

Writing One Character

addch(*ch*)

m_addch(*ch*)

waddch(*win, ch*)

mvaddch(*y, x, ch*)

mvwaddch(*win, y, x, ch*)

The character *ch* is put in the window at the current cursor position of the window. If *ch* is a tab, newline, or backspace, the cursor will be moved appropriately in the window. If *ch* is a different control character, it will be drawn in the ^X notation. The position of the window cursor is advanced. At the right margin, an automatic newline is performed. At the bottom of the scrolling region, if **scrollok** is enabled, the scrolling region will be scrolled up one line.

The *ch* parameter is actually an integer, not a character. Video attributes can be combined with a character by or–ing them into the parameter. This will result in these attributes also being set. (The intent here is that text, including attributes, can be copied from one location to another using **inch** and **addch**.

Writing a String

addstr(*str*)*

m_addstr(*str*)*

waddstr(*win, str*)

mvaddstr(*y, x, str*)

mvwaddstr(*win, y, x, str*)

These functions write all the characters of the null terminated character string *str* on the given window. They are identical to a series of calls to **addch**.

Clearing Areas of the Screen

erase ( )

m_erase ( )

werase(*win*)

These functions copy blanks (spaces) to every position in the window.

erasechar ( )

This function returns an erased character. This is helpful if you erase a character accidentally.

clear( )

m_clear( )

wclear(*win*)

These functions are similar to **erase** and **werase** except that they also call **clearok**, arranging for the screen to be cleared on the next call to **refresh** for that window.

clrtobot( )

wclrtobot(*win*)

These functions erase all lines below the cursor. The current line to the right of the cursor is also erased.

clrtoeol ( )

wclrtoeol(*win*)

These functions erase the current line to the right of the cursor.

Inserting and Deleting Text

delch( )

wdelch(*win*, *c*)

mvdelch(*y*, *x*)

mvwdelch(*win*, *y*, *x*)

The character under the cursor in the window is deleted. All characters to the right on the same line are moved to the left one position. This does not imply use of the hardware delete character feature.

deleteln( )

wdeleteln(*win*)

The line under the cursor in the window is deleted. All lines below the current line are moved up one line. The bottom line of the window is cleared. This does not imply use of the hardware delete line feature.

insch(*c*)

winsch(*win*, *c*)

mvinsch(*y*, *x*, *c*)

mvwinsch(*win*, *y*, *x*, *c*)

The character *c* is inserted before the character under the cursor. All characters to the right are moved one space to the right. The rightmost character on the line may be lost. This does not imply use of the hardware insert character feature.

insertln( )

winsertln(*win*)

A blank line is inserted above the current line. The bottom line is lost.This does not imply use of the hardware insert line feature.

Formatted Output

printw(*fmt*, *arg1*, *arg2*, ...)

wprintw(*win*, *fmt*, *arg1*, *arg2*, . . .)

mvprintw(*y*, *x*, *fmt*, *args*)

mvwprintw(*win*, *y*, *x*, *fmt*, *args*)

These functions correspond to **printf**. **waddch** is being used for the output of characters on the given window instead of **printf**.

Input from a Window

getyx(*win*, *y*, *x*)

The cursor position of the window is placed in the two integer variables *y* and *x*. Since this is a macro, an ampersand (&) is not necessary.

inch( )

winch(*win*)

mvinch(*y*, *x*)

mvwinch(*win*, *y*, *x*)

The character at the current position in the named window is returned. If any attributes are set for that position, their values will be or–ed into the value returned. The predefined constants A_ATTRIBUTES and A_CHARTEXT can be used with the & operator to extract the character or attributes.

Input from the Terminal

getch( )*

wgetch(*win***)**

mvgetch(*y***, *x*)**

mvwgetch(*win***, *y*, *x*)**

A character is read from the terminal associated with the window. In **nodelay** mode, if there is no input waiting, the value –1 is returned. In **delay** mode, the program will hang until the system passes text through to the program. Depending on the setting of **cbreak**, this will be after one character, or after the first newline.

getstr(*str***)**

wgetstr(*win***, *str*)**

mvgetstr(*y***, *x*, *str*)**

mvwgetstr(*win***, *y*, *x*, *str*)**

A series of calls to **getch** is made until a newline is received. The resulting value is placed in the area pointed at by the character pointer *str*. Erase and kill characters are interpreted.

scanw(*fmt***, *arg1*, *arg2*, ...)**

wscanw(*win***, *fmt*, *arg1*, *arg2*, . . .)**

mvscanw(*y***, *x*, *fmt*, *args*)**

mvwscanw(*win***, *y*, *x*, *fmt*, *args*)**

This function corresponds to **scanf**. **wgetstr** is called on the window, and the resulting line is used as input for that scan.

Video Attributes Routines

These functions set the *current attributes* of the named window. These attributes can be any combination of A_STANDOUT, A_REVERSE, A_BOLD, A_DIM, A_BLINK, and A_UNDERLINE. These constants are defined in **<curses.h>** and can be combined with the C | (or) operator.

The current attributes of a window are applied to all characters that are written into the window with **waddch**. Attributes are a property of the character, and move with the character through any scrolling and insert/delete line/character operations.

attroff(*attrs***)***

wattroff(*win***, *attrs*)**

Turns off the named attributes without affecting any other attributes

attron(*attrs***)***

wattron(*win***, *attrs*)**

Turns on the named attributes without affecting any others.

attrset(*attrs***)***

wattrset(*win***, *attrs*)**

Sets the current attributes of the given window to *attrs*.

standout ( )*

wstandout(*win***)**

standout is the same as **attron**(A_STANDOUT).

standend ( )*

wstandend(*win***)**

standend is the same as **attrset**(0), it turns off all attributes.

Bells and Flashing Lights

beep ( )*

flash ( )

These functions are used to signal the programmer. **beep** will sound the audible alarm on the terminal, if possible. **flash** will flash the screen (visible flash), if possible. If neither signal is possible for your current terminal, nothing will happen.

Portability Functions Routines

baudrate ( )*

Queries current terminal and returns the output speed. The number returned is the integer baud rate, for example 9600.

erasechar ( )

The erase character chosen by the user is returned. This is the character typed by the user to erase the character just typed.

killchar ( )

The line kill character chosen by the user is returned. This is the character typed by the user to forget the entire line being typed.

flushinp ( )*

flushinp throws away any **typeahead** that has been typed by the user and has not yet been read by the program.

Cursor Movement Routine

mvcur(*oldrow***, *oldcol*, *newrow*, *newcol*)**

This routine moves the cursor from (*oldrow*, *oldcol*) to (*newrow*, *newcol*).

Miscellaneous Functions Routines

box(*win, vert, hor***)**

Draws a box around the edge of the window. *vert* and *hor* are the characters used to draw the box.

scroll(*win***)**

The window is scrolled up one line. This moves the lines in the window data structure. If the window is *stdscr* and the scrolling region is the entire window, the physical screen will be scrolled at the same time.

delay_output(*ms***)***

Insert pause of *ms* milliseconds in output.

fixterm ( )

Restore terminal to *in curses* state.

flushok (*win, bf***)**

Set the flush–on–refresh flag for *win*.

getcap (*name* **)**

Get terminal capability *name*.

gettmode ( )

Establish current *tty* modes.

has_ic ( )

Has value of **TRUE** if terminal can insert character.

has_il ( )

Has value of **TRUE** if terminal can insert line.

longname ( )

Return verbose name of terminal.

longname(*termbuf, name***)**

Set *name* to the full name of the terminal described by *termbuf*. Used in programs that are compiled with the **–DBSD** option to provide BSD compatibility.

makenew(*window***)**

Sets up a new *window* buffer and returns a pointer to it.

resetterm ( )*

Set *tty* modes to *out of curses* state.

restartterm ( )*

Saves modes and windows prior to restart.

saveterm ( )*

Save current modes as *in curses* state.

setterm(*type***)**

Establish terminal with a give type.

touchline(*win, y, firstcol, numcol***)**

Mark *numcol* columns, starting at column *firstcol*, of line *y* as changed.

touchoverlap(*win1, win2***)**

Mark overlap of *win1* on *win2* as changed.

traceoff ( )

Turn off debugging trace output.

traceon ( )

Turn on debugging trace output.

unctrl(*ch***)***

Use printable version of *ch*.

vsscanf(*buf, fmt, args***)**

Similar to **sscanf** except that it takes a va_list *args* as an argument pointer instead of the argument list itself. This is an internal Curses function.

_showstring(*y, x, first, last, line***)**

Dumps the string running from the string address *first* to the string address *last* out to the terminal on the location (*y, x*). Struct line contains the actual text data. This is an internal Curses function.

Terminfo Level Routines

These routines are called by low–level programs that need access to specific capabilities of **terminfo**. Programs using **terminfo** routines should include both **<curses.h>** and **<term.h>** in that order. If the program needs to use only one terminal, the definition –DSINGLE can be passed to the C compiler. Using this definition can result in a smaller program, but also restricts the program to run on one terminal only. Due to the low level of this interface, its use is discouraged. See Using the Terminfo Level Subroutines for additional information on the use of these routines.

setupterm(*term, fd, rc***)**

Reads in the data base. The *term* parameter is a character string that specifies the terminal name. If *term* is 0, then the value of the TERM environment variable is used. One of the following status values is stored into the integer pointed to by *rc*:

1 Successful completion

0 No such terminal

–1 An error occurred while locating the terminfo data base.

If the *rc* parameter is 0, then no status value is returned, and an error causes **setupterm** to print an error message and exit, rather than return. The *fd* parameter is the file descriptor of the terminal being used for output. **setupterm** calls the **termdef** to determine the number of lines and columns on the display. If **termdef** cannot supply this information, then **setupterm** uses the values in the **terminfo** data base. The simplest call is **setupterm(0,1,0)**, which uses all the defaults.

After the call to **setupterm**, the global variable **cur_term** is set to point to the current structure of terminal capabilities. It is possible for a program to use more than one terminal at a time by calling **setupterm** for each terminal and saving and restoring **cur_term**.

delay_output(*ms***)**

Sets the output delay, in milliseconds.

def_prog_mode

Saves the current terminal mode as program mode, in **cur_term->Nttyb**.

def_shell_mode

Saves the shell mode as normal mode, in **cur_term->Ottyb**. **def_shell_mode** is called automatically by **setupterm**.

putp(*str***)**

Calls **tputs**(*str*, **1**, **putchar**).

reset_prog_mode

Puts the terminal into program mode.

reset_shell_mode

Puts the terminal into shell mode. All programs must call **reset_shell_mode** before they exit. The higher–level routine **endwin** automatically does this.

The **setupterm** subroutine also initializes the global variable **ttytype** as an array of characters to the value of the list of names for the terminal. The list comes from the beginning of the **terminfo** description.

tparm(*str*, *p1*, *p2*, . . . *p9***)**

Instantiates the string *str* with parameters *pi*. The character string returned has the given parameters applied.

tputs(*str*, *affcnt*, *putc***)**

Applies padding information to string *str*. The *affcnt* parameter is the number of lines affected, or 1 if not applicable. The *putc* parameter is a **putchar**–like routine to which the characters are passed one at a time.

Some strings are of a form like $<20>, which is an instruction to pad for 20 milliseconds.

vidputs(*attrs*, *putc***)**

Outputs the string to put terminal in video attribute mode *attrs*. Characters are passed to the **putchar**–like routine *putc*. The *attrs* are defined in **<curses.h>**. The previous mode is retained by this routine

vidattr(*attrs*)

Similar to **vidputs**, but outputs through **putchar**.

Termcap Compatibility Routines

These routines are included for compatibility with programs that require **termcap**. Their parameters are the same as for **termcap**, and they are emulated using the **terminfo** data base.

tgetent(*bp*, *name*) Looks up the **termcap** entry for *name*. Both *bp* and *name* are strings. The *name* parameter is a terminal name; *bp* is ignored. Calls **setupterm**.

tgetflag(*id*) Returns the Boolean entry for *id*, which is a 2–character string that contains a **termcap** identifier.

tgetnum(*id*) Returns the numeric entry for *id*, which is a 2–character string that contains a **termcap** identifier.

tgetstr(*id*, *area*) Returns the string entry for *id*, which is a 2–character string that contains a **termcap** identifier. The *area* parameter is ignored.

tgoto(*cap*, *col*, *row*) Applies parameters to the given *cap*. Calls **tparm**.

tputs(*cap*, *affcnt*, *fn*) Applies padding to *cap* calling *fn* as **putchar**.

two(*ch1*, *ch2*) Make a two letter code into an integer. This is an internal Curses routine used to convert the two letter termcaps into integers so that they can be switched on easily.

twostr(*str*) Makes the first two characters of a string into an integer, similar to **two**. This is also an internal Curses routine.

vsprintf(*buf*, *fmt*, *args*) Uses the format control string specified by the *fmt* parameter to reformat the values specified by the *args* parameter into *buf*. This is an internal Curses routine.

Implementation Specifics

The curses Subroutine Library is part of Base Operating System (BOS) Runtime of AIX for RISC System/6000.

Related Information

The **printf** command, **scanf** command.

The termdef subroutine

The terminfo file format

Function Keys for the curses getch Subroutine

Purpose

Describes the function keys for the **getch** subroutine.

Description

The following function keys might be returned by the **getch** subroutine if **keypad** has been enabled. Note, however, that not all of these are supported due to lack of definitions in **terminfo**, or due to the terminal not transmitting a unique code when the key is pressed.

KEY_BREAK	Break key (unreliable)
KEY_DOWN	Down arrow key
KEY_UP	Up arrow key
KEY_LEFT	Left arrow key
KEY_RIGHT	Right arrow key
KEY_HOME	Home key
KEY_BACKSPACE	Backspace (unreliable)
KEY_F(*n*)	Function key F*n*, where *n* is an integer from 0 to 63
KEY_DL	Delete line
KEY_IL	Insert line
KEY_DC	Delete character
KEY_IC	Insert character or enter insert mode
KEY_EIC	Exit insert character mode
KEY_CLEAR	Clear screen
KEY_EOS	Clear to end of screen
KEY_EOL	Clear to end of line
KEY_SF	Scroll 1 line forward
KEY_SR	Scroll 1 line backwards (reverse)
KEY_NPAGE	Next page
KEY_PPAGE	Previous page
KEY_STAB	Set tab
KEY_CTAB	Clear tab
KEY_CATAB	Clear all tabs

KEY_ENTER	Enter or send (unreliable)
KEY_SRESET	Soft (partial) reset (unreliable)
KEY_RESET	Reset or hard reset (unreliable)
KEY_PRINT	Print or copy
KEY_LL	Home down or bottom (lower left)
KEY_A1	Upper left key of keypad
KEY_A3	Upper right key of keypad
KEY_B2	Center key of keypad
KEY_C1	Lower left key of keypad
KEY_C3	Lower right key of keypad

Implementation Specifics

The curses Subroutine Library is part of Base Operating System (BOS) Runtime of AIX for RISC System/6000.

Related Information

The **curses** subroutine library.

Attributes for Use with Curses Subroutine Library

Purpose

Describes the attributes that can be set using the **curses** subroutine library.

Description

The following video attributes can be passed to the **attron**, **attroff**, and **attrset** subroutines, which are part of the curses library, **libcurses.a**.

A_STANDOUT	The terminal's best highlighting mode
A_UNDERLINE	Underlined
A_REVERSE	Reverse video
A_BLINK	Blinking
A_DIM	Half bright
A_BOLD	Extra bright or bold
A_INVIS	Invisible (blanked or zero–intensity)
A_PROTECT	Protected
A_ALTCHARSET	Alternate character set
A_NORMAL	Normal attributes

Implementation Specifics

The curses Subroutine Library is part of Base Operating System (BOS) Runtime of AIX for RISC System/6000.

Related Information

The **curses** subroutine library.

Screen Attributes Sample Program

The following is a sample code fragment showing how to use the display constants to change the default set of attributes:

```
#include <cur00.h>
#include <cur03.h>

int     attrs[] =
{
        _dBOLD, _dBLINK,
        _dF_WHITE, _dF_RED, _dF_BLUE, _dF_GREEN,
        _dF_BROWN, _dF_MAGENTA, _dF_CYAN, _dF_BLACK,
        _dB_BLACK, _dB_RED, _dB_BLUE, _dB_GREEN,
        _dB_BROWN, _dB_MAGENTA, _dB_CYAN, _dB_WHITE,
        _dREVERSE, _dINVISIBLE, _dDIM, _dUNDERSCORE,
        NULL
};

main( )
{

sel_attr(attrs);
initscr( );
if( REVERSE == NORMAL ) REVERSE = F_BLACK | B_WHITE;
if( INVISIBLE == NORMAL ) INVISIBLE = F_BLACK | B_BLACK;
if( DIM == NORMAL ) DIM = F_BLACK | BOLD;
if( UNDERSCORE == NORMAL ) UNDERSCORE = F_WHITE | B_RED;
STANDOUT = REVERSE;

          <rest of program>

endwin( );
}  /* end main */
```

The routines only define 8 bits of unique attribute information. Selecting foreground color, backrgound color or font requires either 1, 2 or 3 bits depending upon the number of colors or fonts in the list. 1 bit for 2 or fewer, 2 bits for 3 or 4, and 3 bits for 5 to 8. Each character attribute takes 1 bit. However, the attribute names passed to **wcolorout** are variables, so that you can make combinations from the other attributes as shown in the last part of the preceding sample program. If a requested attribute (that is not the terminal default) is equal to NORMAL, then it is either not supported by the terminal, or there is not enough space in the window structure for its mask.

Extended Curses Subroutine Library

Extended Curses Subroutine Library

Purpose

Controls cursor movement and windowing.

Library

Extended Curses Library (**libcur.a**)

Syntax

#include <cur01.h>

Description

The Extended Curses subroutines control input and output to a work station, performing optimized cursor movement, windowing, and other functions. This package is based on the **curses** subroutine package, which is included in most UNIX–compatible systems.

The enhancements provided by Extended Curses include:

- A wider range of display attributes
- Generalized drawing of boxes
- Terminal–independent input data processing
- Extended window control
- Pane, panel, and field concepts
- Support for extended characters
- Handling of mouse input.

To understand how the Extended Curses subroutines can be used in a program, see Curses Programming Example.

For Japanese Language Support: The Extended Curses subroutines also handle the input and display of 2–byte Japanese characters.

Parameters

The following declarations serve for all of the routines:

char *ch* **string*;
NLSCHAR *xc*;
int *line*, *col*, *firstline*, *firstcol*;
int *numlines*, *numcols*, *numchars*, *length*, *mode*;
bool *boolf*;
WINDOW **win*, **win1*, **win2*, **oldwin*, **newwin*;
PANE **pane*;
PANEL **panel*;

Return Values

Unless otherwise noted, each routine returns a value of type **int** that is either **OK** (indicating successful completion) or **ERR** (if an error is encountered).

Header Files

- The **cur00.h** header file replaces **curses.h** when converting programs from the original **curses** package to Extended Curses.
- All routines require the **cur01.h** header file.
- The key codes returned by **getch** are defined in **cur02.h**.
- The **cur03.h** header file defines attribute priority codes, and is not needed by application programs.
- The **unctrl** routine requires **cur04.h**.
- The routines that manage panes and panels (the routines whose names begin with **ec**) also require the **cur05.h** header file.

Naming Conventions

The new routines added to the original **curses** package begin with the letters **ec**.

Many routines operate on **stdscr**, the standard screen, by default. Corresponding routines that allow you to specify a window have the same name, prefixed with the letter **w**. For example, **addch** adds a character to **stdscr**, while **waddch** allows you to specify the window. Sometimes a routine beginning with **p** also exists, such as **paddch**, which allows you to specify a pane.

Some routines also allow you to specify cursor movement with the action to be performed. These routines have a prefix of **mv**. Thus, **addch** becomes **mvaddch**, **waddch** becomes **mvwaddch**, and **paddch** becomes **mvpaddch**. Each of these routines is equivalent to calling **move** or **wmove** before performing the operation.

The various prefixed forms of the routines are implemented as macros. In each case, the routine beginning with **w** is the base subroutine from which the others are defined.

Using the Extended Curses Routines

For ease of use, the **Extended Curses** routines have been separated by function.

- Writing to a Window
- Getting Input from the Terminal
- Controlling the Screen
- Display Attributes

Writing to a Window

Use the following functions to change the contents of a window:

addch (*xc*)
waddch (*win*, *xc*)
waddfld (*win*, *string*, *length*, *numlines*, *numcols*, *mode*, *xc*)

The *xc* parameter is a value of type **NLSCHAR**, rather than a single–byte **char** as used by **curses**.

The **addch** routine adds the **NLSCHAR** specified by the *xc* parameter on the window at the current (*line*, *col*) coordinates. **paddch** adds the character to the presentation space for the pane specified by the *pane* parameter. If the character is '\n' (new–line character), the line is cleared to the end, and the current (*line,col*) coordinates will be changed to the beginning of the next line. A '\r' (return character) moves the current position to the beginning of the current line on the window. A '\t' (tab character) is expanded into spaces in the normal tab stop positions of every eighth column.

Adding a character to the lower right corner of a window that includes the lower right corner of the display causes many terminals to scroll the entire display image up one line. If adding a character or a character attribute causes such scrolling to occur, then **addch** makes the change on the window, but does not mark it for **wrefresh** purposes; **addch** returns the value **ERR**.

Adding only a single–shift control to the window does not change the current position in the window. If the current position in the window does contain only a single–shift control code and *xc* is a valid character data code, then the two are combined to form an IBM National Language Support character, which is added to the window at the current position. Otherwise, *xc* is treated as a valid **NLSCHAR** and is added to the window at the current position.

For Japanese Language Support:

A 2–byte character must be added to **addch** in a single call. If adding a character would cause that character to split across two lines, the system appends a blank to the end of the current line and adds the entire character at the beginning of the following line. If an added character overwrites half an existing 2–byte character, the system replaces the remaining half of that existing character with the partial–character indicator @ (at sign).

The **waddfld** routine adds data to a field within a window. The current coordinates specify the upperleft corner of the field in the window. The *num* and *num* parameters specify the number of lines and columns in the field, respectively. The *length* parameter specifies the length of the data. The *mode* parameter specifies the attribute for the field output. The *xc* parameter specifies the **NLSCHAR** that is used to fill the remainder of the field after the data has been added to it.If the string contains a '\n' (new–line character), the fill character is added to the reminder of the columns on that line of the field, and the remainder of the data is added starting at the first column of the next line of the field. A '\r' (return character) changes the current position to the beginning column of the field. A '\t' (tab character) is expanded with fill characters up to the next normal tabstop position within the field.The **waddfld** routine follows the same rules as **addch** for adding single–shift control codes and character data codes to the window.

For Japanese Language Support:

The fill character must be a 1–byte character. If a 2–byte character is supplied, **waddfld** returns **ERR** and no change is performed.

addstr (*string*)
waddstr (*win*, *string*)
paddstr (*pane*, *string*)
mvaddstr (*line*, *col*, *string*)
mvwaddstr (*win*, *line*, *col*, *string*)
mvpaddstr (*pane*, *line*, *col*, *string*)

The **addstr** routine adds the string pointed to by the *string* parameter on the window at the current (*line*, *col*) coordinates. The string can contain single–shift control codes.

Upon successful completion, **addstr** returns **OK** and the current (*line*, *col*) coordinates point to the location just beyond the end of the string. The **addstr** routine returns **ERR** if an attempt is made to add a character to the lower right corner of a window that includes the lower right corner of the display. In this case, **addstr** writes as much of the string on the window as possible.

waddfld (*win*, *string*, *length*, *numlines*, *numcols*, *mode*, *xc*)

The **waddfld** routine adds data to a field within a window. The current coordinates specify the upperleft corner of the field in the window. The *num* and *num* parameters specify the number of lines and columns in the field, respectively. The *length* parameter specifies the length of the data. The *mode* parameter specifies the attribute for the field output. The *xc* parameter specifies the **NLSCHAR** that is used to fill the remainder of the field after the data has been added to it.

If the string contains a '\n' (new–line character), the fill character is added to the reminder of the columns on that line of the field, and the remainder of the data is added starting at the first column of the next line of the field. A '\r' (return character) changes the current position to the beginning column of the field. A '\t' (tab character) is expanded with fill characters up to the next normal tabstop position within the field.

The **waddfld** routine follows the same rules as **addch** for adding single–shift control codes and character data codes to the window.

For Japanese Language Support:

The fill character must be a 1–byte character. If a 2–byte character is supplied, **waddfld** returns **ERR** and no change is performed.

box (*win*, *vert*, *hor*)
NLSCHAR *vert*, *hor*;

The **box** routine draws a box around the window specified by the *win* parameter. **box** uses the **NLSCHAR** specified by the *vert* parameter to draw the vertical sides of the box, and the **NLSCHAR** specified by the *hor* parameter for drawing the horizontal lines and corners. The *vert* and *hor* parameters must be 1–byte characters.

If the window includes the lower right corner of the display and **scrollok** is not set, then the lower right corner of the box is not shown on the window and the **box** routine returns **ERR**.

The **box** routine is a macro that invokes **superbox**.

cbox (*win*)

The **cbox** routine draws a box around the window specified by the *win* parameter. The characters used are those defined in **/usr/lib/terminfo** or those specified as defaults during initialization. To use the characters defined in **/usr/lib/terminfo**, the application needs to call **wcolorout** prior to the **cbox** routine. Also **wcolorend** afterward.

The **cbox** routine is implemented as a macro that invokes **superbox**.

The **cbox** routine returns **ERR** if the window includes the lower right corner of the display and **scrollok** is not set on.

chgat (*numcharc, mode*)
wchgat (*win, numchars, mode*)
pchgat (*pane, numchars, mode*)
mvchgat (*line, col, numchars, mode*)
mvwchgat (*win, line, col, numchars, mode*)
mvpchgat (*pane, line, col, numchars, mode*)

The **chgat** routine changes the attributes of the next *numchars* characters in the window, starting from the current (*line, col*) coordinates. The attributes are changed to the attributes specified by the *mode* parameter. This routine will not wrap around to the next line; however, specifying a value for the *numchars* parameter that would cause a line wrap is not an error.

The *mode* parameter is one or more of the attributes defined by the global attribute variables. More than one attribute may be specified by logically ORing them together. The following example changes the attributes of the next 10 characters to bold blue characters on a black background:

```
chgat (10, BOLD | F_BLUE | B_BLACK)
```

The **chgat** routine returns **ERR** if the change forces scrolling and **scrollok** is not set to on for the window.

For Japanese Language Support:

chgat (*numcols, mode*)
wchgat (*win, numcols, mode*)
pchgat (*pane, numcols, mode*)
mvchgat (*line, col, numcols, mode*)
mvwchgat (*win, line, col, numcols, mode*)
mvpchgat (*pane, line, col, numcols, mode*)

The **chgat** routine changes the attributes of the next *numcols* columns in the window, starting from the current (*line, col*) coordinates. The attributes are changed to the attributes specified by the *mode* parameter. This routine will not wrap around to the next line; however, specifying a value for the *numcols* parameter that would cause a line wrap is not an error.

Note: The range of columns to be changed should include entire characters. The range should not begin on the second byte of a 2–byte character, nor end on the first byte of a 2–byte character. Beginning or ending a range with one part of a 2–byte character does not cause an error. However, the system does not display a 2–byte character with a different attribute for each byte. Instead, it unpredictably displays such a character with either one attribute or the other.

clear ( )
wclear (*win***)**

The **clear** routine resets the entire **stdscr** (standard screen) window to blank characters. **clear** sets the current (*line*, *col*) coordinates to (0, 0).

clearok (*scr*, *boolf***)**
**WINDOW ****scr*;

The **clearok** routine sets the clear flag for the screen specified by the *scr* parameter. If the *boolf* parameter is **TRUE**, then the screen will be cleared on the next call to **refresh** or **wrefresh**. If the *boolf* parameter is **FALSE**, then the screen will not be cleared on the next call to **refresh** or **wrefresh**. This works only on screens, and, unlike **clear**, does not alter the contents of the screen. If the *scr* parameter is **curscr** (current screen), the next **refresh** will cause a clear–screen sequence, even if the window passed to **refresh** is not a screen.

The **clearok** routine returns **ERR** if the window is not a full screen window.

clrtobot ( )
wclrtobot (*win***)**

The **clrtobot** routine erases the window from the current (*line*, *col*) coordinates to the bottom, leaving the current (*line*, *col*) coordinates unchanged. This does not force a clear–screen sequence on the next refresh.

The **clrtobot** routine always returns the value **OK**.

For Japanese Language Support:

If the current (*line*, *col*) position is on the second byte of a 2–byte character, clearing begins at position *col*–**1**.

clrtoeol ( )
wclrtoeol (*win***)**

The **clrtoeol** routine clears the window from the current (*line*, *col*) coordinates to the end of the current line. The current (*line*, *col*) coordinates are not changed.

The **clrtoeol** routine always returns the value **OK**.

For Japanese Language Support:

If the current (*line*, *col*) position is on the second byte of a 2–byte character, clearing begins at position *col*–**1**.

colorend ( )
wcolorend (*win***)**

The **colorend** routine returns the terminal to **NORMAL** mode. By default, **NORMAL** is usually defined as (**F_WHITE** | **B_BLACK**).

The **colorend** routine is a macro that invokes **xstandend**.

The **colorend** routine always returns the value **OK**.

colorout (*mode*)
wcolorout (*win*, *mode*)

The **colorout** routine sets the current standout bit–pattern of the window (*win*–>**_csbp**) to the attribute specified by the *mode* parameter. Characters added to the window after such a call will have *mode* as their attribute. The *mode* parameter is constructed by logically ORing together attributes that are declared in the **cur01.h** header file that are supported by the terminal.

The **colorout** routine overrides the current setting of the window, and will work in conjunction with almost all of the routines that cause output to be placed on the window.

The **colorout** routine is a macro that invokes **wstandout**.

The **colorout** routine always returns the value **OK**.

delch ( )
wdelch (*win*)
mvdelch (*line*, *col*)
mvwdelch (*win*, *line*, *col*)

The **delch** routine deletes the character at the current (*line*, *col*) coordinates. Each character after the deleted character on the line shifts to the left, and the last character becomes blank.

The **delch** routine always returns the value **OK**.

For Japanese Language Support :

If the current (*line*, *col*) position is on the second byte of a 2–byte character, the position moves back to the first byte before the system deletes that character.

Note: One call to **delch** deletes an entire character, whether it contains 1 or 2 bytes.

deleteln ( )
wdeleteln (*win*)

The **deleteln** routine deletes the current line. Every line below the current line moves up, and the bottom line becomes blank. The current (*line*, *col*) coordinates remain unchanged.

The **deleteln** routine always returns the value **OK**.

drawbox (*win*, *line*, *col*, *numlines*, *numcols*)

The **drawbox** routine draws a box with the upper left corner located at the position specified by the *line* and *col* parameters. The *numlines* parameter specifies the number of rows to be used by the box, and the *numcols* parameter specifies the number of columns to be used by the box.

The characters used to draw the box are either those specified in the **terminfo** file, or those defaulted at initialization.

The **drawbox** routine returns **ERR** if part or all of the box is outside the window, or the box addresses the lower right corner of the screen an **scrollok** is not on.

#include <cur05.h>
ecactp (*pane*, *boolf*)

The **ecactp** routine specifies the active pane in a panel. The pane specified by the *pane* parameter is made the active pane if the *boolf* parameter is **TRUE**. If an active pane has been previously designated, then the border of that pane is reset to the inactive display mode, and the border of the pane specified by the *pane* parameter is set to the active display mode. If the *boolf* parameter is **FALSE**, then the border of the pane specified by the *pane* parameter is set to the inactive display mode.

WINDOW *ecblks ( )

The **ecblks** routine returns a pointer to a window that is filled with blanks. This window is intended to be used as a filler for panes that have no real content. It requires less storage than normal windows because all lines will always contain blanks.

Do not modify or delete this window.

#include <cur05.h>
ecshpl(*panel*)

The **ecshpl** routine shows the panel specified by the *panel* parameter on the terminal.

If the specified panel is currently the top panel, no action is taken and no error is returned. If there is another top panel, the active pane in that panel is changed to the inactive state. The specified panel is placed at the top of the panel chain. This routine should be followed by a call to **ecrfpl** to update the display.

The **ecshpl** routine always returns **OK**.

#include <cur05.h>
ecrfpl(*panel*)

The **ecrfpl** routine refreshes the panel specified by the *panel* parameter. If that panel is partially obscured by other panels, then those panels are also written to the display. If the *panel* parameter is **NULL**, then all panels that have been marked as modified (with **ecpnmodf**) are written. If any panels have been removed (with **ecrmpl**), then all panels are written.

For Japanese Language Support:

If a panel is partially obscured so that half of one or more 2–byte characters is hidden, the system displays the partial–character indicator @ (at sign) in place of the visible half of the character or characters. This display in no way affects the data stored for the panel. If the obscured part of the panel is later uncovered, the system again displays the full 2–byte character.

#include <cur05.h>
ecrfpn(*pane*)

The **ecrfpn** routine refreshes the pane specified by the *pane* parameter on the display. If the pane is the active pane, then the window might be scrolled to ensure that the cursor is visible. If the pane is not active, then the window is not scrolled.

The **ecrfpn** routine always returns **OK**.

#include <cur05.h>
ecrmpl(*panel*)

The **ecrmpl** routine removes the panel specified by the *panel* parameter from the list of panels that are currently being displayed. If the panel is not currently in that list, no action is taken and no error is returned. This routine should be followed by a call to **ecrfpl** to update the display.

The **ecrmpl** routine always returns **OK**.

#include <cur05.h>
ecscpn(*pane*, *numlines*, *numcols*)

The **ecscpn** routine causes the pane specified by the *pane* parameter to be scrolled over the underlying window the distance indicated by the *numcols* and the *numlines* parameters. The *numcols* parameter specifies the distance to scroll horizontally and the *numlines* parameter specifies the distance to scroll vertically. These parameters can be positive or negative and can imply a movement that positions the viewport partially or completely off the window. If such a position results from the scroll, the scroll stops after moving as far in the indicated direction as possible. Positive values move to the right or down. Negative values move to the left or up.

If there are other panes linked to the pane specified, those panes will also scroll an amount necessary to maintain the identical horizontal or vertical positioning on the respective windows. If the resulting position requires placing the viewport partially or completely off the window, the scroll request terminates at the edge of the window.

erase( )
werase(*win*)
perase(*pane*)

The **erase** routine clears the window and sets it to blanks without setting the clear flag. Similarly, **perase** erases the pane specified by the *pane* parameter. This is analogous to the **clear** routine, except that it does not cause a clear–screen sequence to be generated on a **refresh**.

fullbox(*win*, *vert*, *hor*, *topl*, *topr*, *botl*, *botr*)
NLSCHAR *vert*, *hor*, *topl*, *topr*, *botl*, *botr*;

The **fullbox** routine puts box characters on the edges of the window. The *vert* parameter specifies the **NLSCHAR** to use for the vertical sides. The *hor* parameter specifies the **NLSCHAR** to use for the horizontal lines. The *topl* and the *topr* parameters specify the **NLSCHAR**s to use for the top left and the top right corners. The *botl* and the *botr* parameters specify the **NLSCHAR**s to use for the bottom left and the bottom right corners.

For Japanese Language Support:

The **fullbox** routine does not accept 2–byte box characters. If a 2–byte character is used, the **fullbox** routine substitutes a 1–byte character and draws the box. The system returns **ERR**.

The **fullbox** routine returns **ERR** if an attempt is made to scroll when **scrollok** is not active.

The **fullbox** routine is a macro that invokes **superbox**.

insch (*xc*)
winsch (*win*, *xc*)
mvwinsch (*win*, *line*, *col*, *xc*)
mvinsch (*line*, *col*, *xc*)

The **insch** routine inserts the **NLSCHAR** specified by the *xc* parameter into the window at the current (*line*, *col*) coordinates. Each character after the inserted character shifts to the right, and the last byte on the line disappears.

For Japanese Language Support:.

If the current position is at the second byte of a 2–byte character, the position is moved left to the first byte of that character before the specified **NLCHAR** is inserted.

The **insch** routine always returns the value **OK**.

insertln ( )
winsertln (*win*)

The **insertln** routine inserts a line above the current line. Each line below the current line is shifted down, and the bottom line disappears. The current line becomes blank and the current (*line*, *col*) coordinates remain unchanged.

The **insertln** routine always returns the value **OK**.

move (*line*, *col*)
wmove (*win*, *line*, *col*)

The **move** routine changes the current (*line*, *col*) coordinates of the window to the coordinates specified by the *line* and *col* parameters.

The **move** routine returns **ERR** if the destination for the cursor is outside the window or viewport.

overlay (*win1*, *win2*)

The **overlay** routine overlays the window specified by the *win1* parameter on the window specified by the *win2* parameter. The contents of the window specified by the *win1* parameter, insofar as they fit, are placed on the window specified by the *win2* parameter at their starting (*line*, *col*) coordinates. This is done nondestructively; that is, blanks on the *win1* window leave the contents of the space on the *win2* window untouched.

The **overlay** routine moves data only if the data is nonblank or if the display attribute is different.

The only data that is considered for moving from the *win1* window to the *win2* window is data that occupies display positions that are common to both windows.

The **overlay** routine is implemented as a macro that invokes **overput**, which uses **waddch** to transfer the data from window to window.

The **overlay** routine returns **ERR** if the overlay addresses the lower right corner of the display and **scrollok** is **FALSE**.

overwrite (*win1*, *win2*)

The **overwrite** routine copies data from the window specified by the *win1* parameter to the window specified by the *win2* parameter. The contents of the *win1* window, insofar as they fit, are placed on the *win2* window at their starting (*line*, *col*) coordinates. This is done destructively; that is, blanks on the *win1* window become blanks on the *win2* window.

Only the data that occupies positions on the display that are common to the two windows will be moved from the *win1* window to the *win2* window.

The **overwrite** routine is implemented as a macro that invokes **overput** which uses **waddch** to transfer the data from window to window.

The **overwrite** routine returns **ERR** if an attempt is made to write to the lower right corner and **scrollok** is not set.

printw (*fmt* [, *value*, . . .])
wprintw (*win*, *fmt* [, *value*, . . .])
char **fmt*;

The **printw** routine performs a **printf** on the window using the format control string specified by the *fmt* parameter and the values specified by the *value* parameters. The output to the window starts at the current (*line*, *col*) coordinates. Use the field width options of **printf** to avoid leaving items on the window from earlier calls.

Note: The maximum length of the format control string after being expanded is 512 characters.

The **printw** routine returns **ERR** if it causes the screen to scroll illegally.

refresh ( )
wrefresh (*win*)

The **refresh** routine synchronizes the terminal screen with the window. If the window is not a screen, then only the part of the display covered by it is updated. **refresh** checks for possible scroll errors at display time.

The **refresh** routine returns **ERR** if the change specified is in the last position of a window that includes the lower right corner of the display, or if they would cause the screen to scroll illegally. If they would cause the screen to scroll illegally, **refresh** updates whatever can be updated without causing the scroll.

standend ( )
wstandend (*win*)

The **standend** routine stops displaying characters in standout mode.

standout ( )
wstandout (*win*)

The **standout** routine starts displaying characters in standout mode. Any characters added to the window are put in standout mode on the terminal if the terminal has that capability. The first available attribute as determined by **sel_attr** is used for standout. This is normally the reverse attribute when the default display attribute priority is used.

The **standout** routine always returns the value **OK**.

Use the **refresh** routine to transfer the contents of the current window to the screen after all changes to the window are complete. The **refresh** routine does not rewrite any part of the window that has not changed since the last refresh call. To force the whole window to be rewritten, use the **touchwin** routine before the **refresh** routine. Also use **ecrfpn** to refresh a pane, and **ecrfpl** to refresh a panel.

superbox (*win*, *line*, *col*, *numlines*, *numcols*, *vert*, *hor*, *topl*, *topr*, *botl*, *botr*)
NLSCHAR *vert*, *hor*, *topl*, *topr*, *botl*, *botr*;

The **superbox** routine draws a box on the window specified by the *win* parameter. The *line* and *col* parameters specify the starting coordinates for the box. The *numlines* parameter specifies the depth of the box. The *numcols* parameter specifies the width of the box. The *vert* parameter specifies the **NLSCHAR** to use for vertical delimiting. The *hor* parameter specifies the **NLSCHAR** to use for horizontal delimiting. The *topl*, *topr*, *botl*, and *botr* parameters specify the **NLSCHAR**s to use for the top left corner, the top right corner, the bottom left corner, and the bottom right corner, respectively.

If the window specified by the *win* parameter is a **_SCROLLWIN** window and scrolling is not allowed, then the bottom right corner is not put on the window.

The **superbox** routine uses **addch** to place the characters on the window.

For Japanese Language Support:

If any of the box characters is a 2–byte character, the **superbox** routine substitutes a 1–byte box character for every 2–byte box character. The box is drawn with the 1–byte box characters, but **superbox** returns **ERR**.

The **superbox** routine returns **ERR** if the defined box is outside the window, or an attempt is made to write to the lower right corner of the display when **scrollok** is off.

Getting Input from the Terminal

Input is the complementary function to output. The screen package needs to know what is on the terminal at all times. Therefore, if a program echoes input characters, the terminal must be in a mode that passes characters immediately to the program, rather than waiting for a carriage return to send input to the program. The **getch** routine sets the terminal to the character input mode and then reads in the character.

Use the following routines for input from the terminal:

crmode ( )
nocrmode ( )

The **crmode** routine turns off the canonical processing of input by the system device driver. When canonical processing is off, data is made available without waiting for a '\n' (new–line character). The **nocrmode** routine enables canonical processing by the system device driver.

The **wgetch** routine, which is used for all Extended Curses input, forces the equivalent of **crmode** before requesting input if echoing is active, and reinstates the original status on exit. If you are using echo, you should issue a call to either **crmode** or **raw** to avoid multiple calls by **wgetch**.

The **crmode** routine differs from **raw** in that **crmode** has no effect on output data processing and does not disable signal processing by the device driver.

The **crmode** routine always returns the value **OK**.

cresetty (*boolf*)

The **cresetty** routine resets the terminal to the state saved by the last call to **csavetty**. Use this routine after the completion of a program that uses the terminal as a simple terminal. If the *boolf* parameter is **TRUE**, then the data in **curscr** is redisplayed.

csavetty (*boolf*)

The **csavetty** routine saves the current Extended Curses state so that it can later be reset by **cresetty**. Use this routine before running a program that uses the terminal as a simple terminal. If the *boolf* parameter is **TRUE**, then the following status is set before saving the terminal status: **crmode**, **noecho**, **meta**, **nonl**, and **keypad** (TRUE).

echo( )
noecho( )

The **echo** routine causes the terminal to echo characters to the display. If **echo** is set on, **wgetch** places all input into the data structure for the window.

The **noecho** routine turns **echo** off. If **echo** is turned off, characters are not written to the display.

#include <cur05.h>
ecflin (*pane*, *firstline*, *firstcol*, *numlines*, *numcols*, *pat*, *xc*, *buf*, *mask*)
NLecflin (*pane*, *firstline*, *firstcol*, *numlines*, *numcols*, *pat*, *xc*, *buf*, *length*, *mask*)
char **pat*, **buf*, **mask*;

The **ecflin** and **NLecflin** routines input field data to a pane. **NLecflin** is supplied for international character support, and **ecflin** is retained to preserve traditional functionality. **NLecflin** works like **ecflin**, but has an additional parameter, *length*, which specifies the length of the buffer in which the input data is stored.

The **ecflin** routine inputs field data to the pane pointed to by the *pane* parameter. The *firstline* and the *firstcol* parameters specify the upper left corner of the field in the current window being shown in the pane. The *numcols* parameter specifies the number of columns in the field, and the *numlines* parameter specifies the number of rows in the field.

The *buf* parameter points to a buffer in which input data is stored. This buffer must be at least *numlines* x *numcols* characters long.

Note: When **ecflin** is called, *buf* should contain data representing the initial contents of the field, with an uninitialized field represented by null characters. If **ecflin** is called to operate on a field in which data is already displayed, *buf* should reflect that data as though the field contained the output of a previous call to **ecflin**. The characters in *buf* are edited to correspond to changes in the displayed field.

The *xc* parameter specifies the first **NLSCHAR** to be entered in the field. If the *xc* parameter is a null character, it is ignored.

The *pat* and *mask* parameters specify the set of characters that are to be accepted as valid input.

The position in the field may not always correspond to the position in the input buffer, since a 2–byte extended character corresponds to a single display character in the field in the window. Input is accepted from the terminal as long as the cursor remains within the bounds of the field. However, if the input buffer is filled before the cursor exits the field, input processing stops and **ecflin** returns.

Cursor movement that moves the cursor outside the field is allowed and is reflected on the display. If cursor movement places the cursor in a position where data input would cause the input buffer to overflow, input processing stops. Any data keys entered are checked against the character set specified by the *pat* parameter. If the data character is acceptable, then it is echoed. If the character is not acceptable, then the **ecflin** routine returns its value.

Insert and delete keys are honored and data are shifted within the field as needed. If the field spans more than one line and insertions or deletions are made, then data that are shifted out of one line of the field are shifted into the end of the next line. Data shifted out of the field are lost. When characters are deleted, null characters are shifted into the end of the field.

For Japanese Language Support:

Cursor motion always leaves the cursor on the first byte of a 2–byte character. If the cursor is moved vertically in such a way that it rests on the second byte of a character, the system moves the cursor to the left, placing it on the first byte of the character.

When an entered character overwrites half an existing 2–byte character, the system replaces the remaining half of that existing character with a partial–character indicator @ (at sign). Attempting to insert a 2–byte character in the last column of the last line also causes the system to insert an @. If an insertion shifts a 2–byte character so that it begins in the last column of the last line, the system again places an @ in that position. In all such cases, the partial–character indicator appears both on the screen and in *buf*.

When a 2–byte character is entered in the last column of any line but the last, or when shifting as a result of insertion would cause a 2–byte character to begin in the last column of a line, the system temporarily splits the character. The screen then displays partial–character indicators in both the last column of the line and the first column of the following line.

If subsequent editing moves the split 2–byte character so that it appears on only one line, the system again correctly displays the full character. However, if the character remains split when **ecflin** returns, the partial–character indicators remain displayed, and the two parts of the character cannot subsequently be rejoined. In such a case, the original character value is lost, but the contents of *buf* continue to show the actual character rather than partial–character indicators.

The *pat* parameter points to a string that indicates the set of characters that is acceptable as valid input. These characters include all code points of the P0, P1, and P2 code pages (see dispsym.).

For Japanese Language Support:

The valid set of input characters includes all 1–byte code points, plus a string that represents 2–byte codes from specific groups.

The string is formed from the following codes:

U Uppercase letters: 'A—Z', also uses the accented uppercase letters from code pages P0, P1, and P2.

L Lowercase letters: 'a—z', also uses the accented lowercase letters from code pages P0, P1, and P2.

N Numeric characters: '0—9', also uses 2–byte codes.

A Alphanumeric characters: 'A—Z', 'a—z', and '0—9', also uses the accented letters from code pages P0, P1, and P2.

B Blank (space character—0x20).

P Printable characters: *blank*—'~' (0x20—0x7E).

G Graphic characters: '!'—'~' (0x21—0x7E)

X Hexadecimal characters: '0—9', 'A—F', and 'a—f'.

Note: The codes U, L, A, B, P, G, and X allow only single–byte ASCII codes in order to maintain compatibility with previous versions of **ecflin**.

C Control Characters:

- Cursor Up, Cursor Down, Cursor Left, Cursor Right
- Backspace
- Back–tab (to first position of field)
- Insert (enable or disable insert mode)
- Delete (delete current character)
- New–line (to left column and down one line).

D Default characters:

- 0x20—0x7E
- 0x80—0xFF
- 0x1FA0—0x1FFF
- 0x1E80—0x1EFF
- 0x1DA0—0x1DFF
- 0x1C80—0x1CFF
- Controls, as defined for code **C**.

Note: Allowing control characters (code **C**) means the keypad can be used to move the cursor, select insert mode, and so on. The codes for control keys are not returned as field input.

For Japanese Language Support:

D Default characters:

- All ASCII graphic characters
- All 2–byte codes
- Controls, as defined for code **C**.

J All Japanese text characters, including 1– and 2–byte katakana, hiragana, and kanjii.

H Hiragana.

k One–byte katakana.

K All katakana (1– and 2–byte forms).

These codes can be combined. For example, `"HK–k"` allows all 2–byte kana characters (hiragana and 2–byte katakana).

Note: The codes U, L, A, B, P, G, and X allow only single–byte ASCII codes in order to maintain compatibility with previous versions of **ecflin**.

Z Application–specified character set.

+ Allows characters indicated by following codes.

– Does not allow characters indicated by the following codes.

If the first character of *pat* is + or –, the set of characters specified by the rest of the string is added to (+) or taken from (–) the default characters (which can also be specified with **D**). If the first character in this string is not + or –, the set of characters specified by *pat* replaces the default. After the first character, the sets indicated are allowed unless preceded by a – (minus or hyphen). For example:

`"PC–L"` Allows the printable and control characters, except for lowercase letters.

`"–CBN"` Allows all of the default characters, except for control characters, blanks, or numeric characters.

If the *pat* string contains a **Z**, the array to which the *mask* parameter points specifies a character validity mask. This array must be exactly 128 bytes long (1024 bits), where each bit corresponds to a character codes as returned by the **wgetch** routine.

The bytes in the array correspond as follows:

Bytes	Characters Selected
0–31	P0 characters 0x00–0xFF
32–63	Keycodes 0x100–0x1FF
64–79	Low P1 characters 0x1F80–0x1FFF
80–95	High P1 characters 0x1E80–0x1EFF
96–111	Low P2 characters 0x1D80–0x1DFF
112–127	High P2 characters 0x1C80–0x1CFF

For Japanese Language Support:

If the *pat* string contains a **Z**, the array to which the *mask* parameter points specifies a character validity mask. This array must be exactly 64 bytes long (512 bits). Within the bytes of *mask*, the upper bit corresponds to the first code.

Because of the many characters used in Japanese Language Support, a complete bit map of the character set would be so large as to be unamangeable. Instead of a bit map, Japanese Language Support provides separate mask bits that are used for 1–byte codes. You can select 2–byte codes in groups based on the first byte of their encoding.

The bytes in the array correspond as follows:

Bytes	Characters Selected
0–15	ASCII subset, 0x00–0x7F
16	Special (see following list)
17–19	Kanjii, 0x88??–0x9F??
20–27	One–byte katakana, codes 0xA0–0xDF
28–31	Kanjii conitnued, codes 0xE0??–0xFF??
32–37	Keypad special keys
38–55	(Reserved)
56–63	Function keys

With the exception of the bits in byte 16, a character's bit position in *mask* corresponds in the following way:

- One–byte character Bit position in *mask* corresponds to its code.
- Two–byte character Bit position in *mask* corresponds to the upper byte of its code.

The values in byte 16 represent codes 0x81??–0x84??. These codes are divided because the upper byte is not sufficient to create logical groupings of characters. The bit values within byte 16 are as follows:

Value	Codes	Characters Selected
0x80	0x81??	Punctuation and symbols
0x40	0x824F–0x8258	Two–byte digits
0x20	0x8260–0x829E	Two–byte roman alphabet
0x10	0x829F–0x82FF	Hiragana
0x08	0x8340–0x839E	Two–byte katakana
0x04	0x839F–0x83FF	Greek alphabet
0x02	0x8440–0x849E	Cyrillic alphabet
0x01	0x849F–0x84FF	Line drawing

If a given bit is set to 1, the corresponding character is accepted (for **+Z**) or rejected (for **–Z**). If a bit is set to 0, then the acceptance status of the corresponding character, as determined by the rest of *pat*, is not changed.

Upon successful completion, the **ecflin** routine returns the code associated with the last input that terminated an operation.

The **ecflin** routine returns **ERR** if one or more of the following are true:

- There is an error in the parameters.
- The *firstline* parameter is outside the window.
- The *firstcol* parameter is outside the window.
- The *numcols* parameter is too large.
- The *numlines* parameter is too large.

#include <cur05.h>
ecpnin(*pane*, *boolf*, *xc***)**

The **ecpnin** routine causes the pane to accept keyboard input. The pane specified by the *pane* parameter is scrolled, if necessary, to ensure that the cursor is visible on the display. Keyboard input is then accepted. If the *boolf* parameter is **TRUE** and if the input character is a simple cursor movement, then the resulting cursor position is reflected on the display. Further input is then read from the terminal. If the *boolf* parameter is **FALSE**, or if the input character is not a simple cursor movement, then the value of the input character is returned.

The *xc* parameter specifies the first **NLSCHAR** to be assumed from the display. If *xc* is a null character, then it is ignored.

This routine tracks the mouse cursor if mouse tracking is enabled.

extended(*boolf***)**

The **extended** routine turns on and off the combining of input bytes into 2–byte extended characters. If the *boolf* parameter is **TRUE**, then this input processing is turned on; if **FALSE**, then it is turned off. By default, **extended** processing is initially turned on.

#include <cur02.h>
int getch( )
int wgetch(*win***)**
int mvgetch(*line*, *col***)**
int mvwgetch(*win*, *line*, *col***)**

The **getch** routine gets a character from the terminal and echoes it on the window, if necessary. If **noecho** has been set, then the window does not change. The **noecho** routine and either **crmode** or **raw** must be set for Extended Curses to know what is actually on the terminal. If these settings are not correct, **wgetch** sets **noecho** and **crmode** and resets them to the original mode when done.

If **extended** processing is turned on, then **getch** combines input sequences that contain single–shift controls into 2–byte extended characters. By default, **extended** processing is turned on.

Upon completion, the character code for the data key or one of the following values is returned:

KEY_NOKEY	**nodelay** is active and no data is available.
KEY_*xxxx*	**keypad** is active and a control key was recognized. See the **cur02.h** header file for a complete list of the key codes that can be returned.
ERR	Echoing the character would cause the screen to scroll illegally.

#include <cur02.h>
int getstr (*string***)**
int wgetstr (*win*, *string***)**
int mvgetstr (*line*, *col*, *string***)**
int mvwgetstr (*win*, *line*, *col*, *string***)**

The **getstr** routine gets a string through the window and stores it in the location pointed to by the *string* parameter. The string can contain single–shift control codes. The area pointed to must be large enough to hold the string. The **getstr** routine calls **wgetch** to get the characters until a new–line character or some other control character is encountered.

For Japanese Language Support:

The string cannot contain single–shift codes.

Upon completion, one of the following values is returned:

OK	The input string was terminated with a new–line character.
KEY_NOKEY	**nodelay** is active and no data is available.
KEY_*xxxx*	The input string ended with a control key, and the code for this key was returned. See the **cur02.h** header file for a complete list of the key codes that can be returned.
ERR	The string caused the screen to scroll illegally.

keypad (*boolf*)

The **keypad** routine turns on and off the mapping of key sequences to single integers. If the *boolf* parameter is **TRUE**, input processing is turned on. If the *boolf* parameter is **FALSE**, input processing is turned off. By default, input processing is initially turned off.

When turned on, sequences of characters from the terminal are translated into integers that are defined in the **cur02.h** header file.

The codes available on a given terminal are determined by the **terminfo** terminal description file.

The **keypad** routine always returns the value **OK**.

raw ( )
noraw ( )

The **raw** routine sets the terminal to raw mode. In raw mode, canonical processing by the device driver and signal processing are turned off. The **noraw** routine turns off raw mode.

The **raw** and **noraw** routines always return the value **OK**.

Controlling the Screen

Use the following library routines to control and manipulate the windows, panes, and panels on the screen:

delwin (*win*)

The **delwin** routine deletes the window specified by the*win* parameter. All resources used by the deleted window are freed for future use.

If a window has a subwindow allocated inside of it, the deletion of the window does not affect the subwindow even though the subwindow is invalidated. Therefore, subwindows must be deleted before the outer windows are deleted.

The **delwin** routine always returns the value **OK**.

#include <cur05.h>
ecadpn (*pane*, *win*)

The **ecadpn** routine adds the window specified by the *win* parameter to the list of windows that can be presented in the pane specified by the *pane* parameter. No visible action occurs as a result of this routine. A call to **ecaspn** must be made after **ecadpn** to change the data associated with the pane display.

The **ecadpn** routine returns **ERR** if the system is unable to allocate the storage required.

#include <cur05.h>
ecaspn (*pane*, *win*)

The **ecaspn** routine makes the window specified by the *win* parameter the current window for display in the pane specified by the *pane* parameter. A refresh call for the pane or panel is needed to cause the data to be presented on the display. The viewport associated with the pane is positioned with the top left corner of the viewport at the top left corner of the data for the window.

The **ecaspn** routine returns **ERR** if the window specified by the *win* parameter was not previously associated with this pane using **ecadpn**.

#include <cur05.h>
PANEL *ecbpls (*numlines*, *numcols*, *firstline*, *firstcol*, *title*, *divdim*, *border*, *pane*)
short *numlines*, *numcols*, *firstline*, *firstco;*;
char **title*;
char *divdim*, *border*;

The **ecbpls** routine builds a panel structure.

The *numlines* parameter specifies the panel size in rows.

The *numcols* parameter specifies the panel size in columns.

The *firstline* parameter specifies the panel's origin on the display's upper left corner row coordinate.

The *firstcol* parameter specifies the panel's origin on the display's upper left corner column coordinate.

The *title* parameter points to a title string. The title is shown centered in the top border. If no title is desired, this parameter should be NULL.

The *divdim* parameter specifies the dimension along which this panel is to be divided: either **Pdivtyv** (vertical) or **Pdivtyh** (horizontal).

The *border* parameter indicates whether or not this panel is to have a border: either **Pbordry** (yes) or **Pbordrn** (no).

The *pane* parameter points to the first pane that defines the divisions of this panel.

All parameters should be given as defined here. However, they are not checked or used until a call is made to **ecdvpl**. An application may modify values put into this structure until it calls **ecdvpl**.

Upon successful completion, a pointer to the new panel is returned. **ecbpls** returns **ERR** if there is not enough storage available.

#include <cur05.h>
PANE *ecbpns (*numlines*, *numcols*, *ln*, *ld*, *divdim*, *ds*, *du*, *border*, *lh*, *lv*)
short *numlines*, *numcols*, *ds*;
PANE **ln*, **ld*, **lh*, **lv*;
char *divdim*, *du*, *border*;

The **ecbpns** routine builds a pane structure.

The *numlines* parameter specifies the number of rows in the presentation space for the pane.

The *numcols* parameter specifies the number of columns in the presentation space for the pane.

The *ln* parameter points to a neighboring pane either above or to the left.

The *ld* parameter points to the start of a chain for divisions of the pane.

The *divdim* parameter specifies the dimension of the pane along which division is to occur. This parameter is used if and only if the *ld* parameter is not **NULL**. Valid values for this parameter are **Pdivpnv** (vertical dimension) and **Pdivpnh** (horizontal dimension).

The *ds* and *du* parameters together specify the size of this pane as part of the division of a parent pane:

du	**Vertical or Horizontal Size of the Pane**
Pdivszc	The size is specified by the *ds* parameter.
Pdivszp	The size is *ds* divided by 10000 of the available space. For example, if *ds* is 5000, then the row or column size is half of the available space.
Pdivszf	The pane has a floating size. The value of the *ds* parameter is not used.

The *border* parameter specifies whether or not this pane has a border: either **Pbordry** (yes) or **Pbordrn** (no).

The *lh* parameter points to a pane that is to scroll with this pane when the pane scrolls horizontally.

The *lv* parameter points to a pane that is to scroll with this pane when the pane scrolls vertically.

If you specify **NULL** for the *ld* parameter, or if you are not sure which value to use for *du*, then specify **Pdivszf** for the *du* parameter.

If the *ln* parameter is not **NULL**, the **divs** field of the pane structure being built receives the value that was in the *ln*.**divs** field. The *ln*.**divs** field is modified to point to the new pane structure being built.

If the *lh* and the *lv* parameters are not **NULL**, they will be used to link the new structure to the specified structures and to link the specified structures to the new structure. The links thus created will form a ring that includes all panes that scroll together.

Upon successful completion, a pointer to the new pane structure is returned. **ecbpns** returns **ERR** if a error is detected during processing.

#include <cur05.h>
ecdfpl (*panel*, *boolf*)

The **ecdfpl** routine creates the Extended Curses window structures needed to define the specified panel.

At the time this routine is invoked, all size and location specifications of the panel and its constituent panes must be properly set. **ecdfpl** does not examine any of the division size specifications or the scroll link specifications.

The **fpane** pointer in the indicated panel structure must point to the first leaf pane for the panel, and the subsequent **nxtpn** pointers from that pane must form a loop back to the first leaf pane. (This is done by **ecdvpl**.)

A **WINDOW** structure is built for the panel specified by the panel parameter. This **WINDOW** will have a size that corresponds to the size of the panel. For each of the panes in the subsequent chain, a separate **WINDOW** structure is built with a size that corresponds to the specified presentation space size or the viewport size, whichever is larger.

If borders are specified for any of the panes, those borders are drawn on the **WINDOW** for the panel. All corners are checked and, if needed, proper junction characters are used to draw the corner.

The *boolf* parameter indicates whether to suppress the creation of presentation spaces for the panes. If the value is **TRUE**, then presentation spaces are not created. If **FALSE**, then presentation spaces are created.

The **ecdfpl** routine returns **ERR** if sufficient storage is not available for the **WINDOW** structures being created.

#include <cur05.h>
ecdppn (*pane*, *oldwin*, *newwin*)

The **ecdppn** routine adds, drops or replaces a presentation space for a pane.

First, if the *oldwin* parameter is not **NULL**, then **ecdppn** drops *oldwin* from the list of windows that are alternatives for the pane specified by the *pane* parameter. The previous association should have been established using **ecadpn**. If the *oldwin* parameter is **NULL**, then no window is dropped.

Next, if the *newwin* parameter is not **NULL**, then **ecdppn** adds *newwin* as a valid pane for this window, replacing *oldwin*, if it was associated with the pane specified by the *pane* parameter. (See **ecadpn** for a better way to add a pane.)

The **ecdppn** routine always returns **OK**.

#include <cur05.h>
ecdspl (*panel*)

The **ecdspl** routine releases all of the data structures associated with the panel specified by the *panel* parameter. The released data structures are returned to the free pool. The released data structures include the panel structure, all associated pane structures, any window structures associated with the panes, any auxiliary window structures associated with the panes, and all private control structures used by Extended Curses.

#include <cur05.h>
ecdvpl (*panel*)

The **ecdvpl** routine assigns a real size and relative position to all the panes defined for the panel specified by the *panel* parameter. All of the panes must be linked to the panel. The structure of a tree will be followed to determine the sizes for each pane.

The direction of the first set of divisions and the size of the first set of divisions is determined. This information is used to control the division algorithm. First, the total space for the interior of panes is determined by counting the panes and their borders, using the size along the direction of division. Next, any panes with *fixed* size are given the space indicated by the **divsz** field in the pane structure. The remaining available space is then assigned to the panes that have specified a *proportional* size. Finally, any space that remains is assigned to those panes that specified a *floating* size. Once the sizes are determined, the origin for each pane relative to the panel origin is determined and entered into the pane structure. A final pass is made over the list of panes in the current division, and, for each that is itself divided, the process is repeated.

If adjacent panes both have a border specified, the border space is shared between them.

If all of the panes have a fixed size and the total is less than the available space, there will be space that cannot be accessed by the application in the resulting structure.

If, after allocating space to the proportional panes, there is space remaining and no floating panes are in the current set, the remaining free space is allocated to the proportional panes.

The **ecdvpl** routine returns **ERR** and the structures are invalid for use by **ecdfpl** if one or more of the following occur:

void ecpnmodf(*pane***)**

The **ecpnmodf** macro marks the panel that contains the pane specified by the *pane* parameter as modified. This information is used by **ecrfpl** to determine whether a panel needs to be written to the display.

- The total size specified for fixed panes exceeds the space available.
- The total fractions specified for the proportional panes exceed a total of 1.
- The number of panes exceeds the number of positions available.

#include <cur05.h>
ecrlpl(*panel***)**

The **ecrlpl** routine returns the structures associated with the panel specified by the *panel* parameter to the free storage pool. This includes all window structures associated with the panes of the panel, all Extended Curses private structures, and any added window structures. The panel and associated pane structures are not released and can be reused.

The **ecrlpl** routine always returns **OK**.

#include <cur05.h>
ectitl(*title*, *line*, *col***)**
char **title*;

The **ectitl** routine creates or modifies the title panel. The title panel is always visible, that is, on top of any other panels. The *title* parameter points to a character string that is displayed as the new title. If *title* is **NULL**, then any existing title is removed. The *line* and *col* parameters specify the coordinates for the upper left corner of the title panel. If *firstline* is not valid, then it defaults to 1. If *firstcol* is not valid, then the title will be centered.

endwin( )

The **endwin** routine ends window routines before exiting. Ending window routines before exiting restores the terminal to the state it was before **initscr** (or **gettmode** and **setterm**) was called. **endwin** should always be called before exiting. **endwin** does not exit.

gettmode ( )

The **gettmode** routine issues the needed control operation to the display device driver to save the processing flags in a fixed global area. **gettmode** is invoked by **initscr** and is not normally called directly by applications.

getyx (*win*, *line*, *col***)**

The **getyx** routine stores the current (*line*, *col*) coordinates of the window specified by the *win* parameter into the variables *line* and *col*. Because **getyx** is a macro and not a subroutine, the names of *line* and *col* passed, not their addresses.

Upon successful completion, *line* and *col* contain the current row and column coordinates for the cursor in the specified window.

NLSCHAR inch ( )
NLSCHAR winch (*win*)
NLSCHAR mvinch (*line*, *col*)
NLSCHAR mvwinch (*win*, *line*, *col*)

The **inch** routine returns the **NLSCHAR** at the current (*line*, *col*) coordinates on the specified window. No changes are made to the window.

Upon successful completion, the code for the character located at the current cursor location is returned.

WINDOW *initscr ( )

The **initscr** routine performs screen initialization. **initscr** must be called before any of the screen routines are used. It initializes the terminal–type data, and without it, none of the Extended Curses routines can operate properly.

If standard input is not a **tty**, **initscr** sets the specifications to the terminal whose name is pointed to by **Def_term** (initially "dumb"). If the value of the **bool** global variable **My_term** is **TRUE**, **Def_term** is always used.

If standard input is a terminal, the specifications for the terminal named in the environment variable **TERM** are used. These specifications are obtained from the **terminfo** description file for that terminal.

The **initscr** routine creates the structures for **stdscr**, (the standard screen) and **curscr** (the current screen) and saves the pointers to those structures in global variables with the corresponding names.

Upon successful completion, a pointer to **stdscr** is returned.

leaveok (*win*, *boolf*)

The **leaveok** routine sets a flag, used by the window specified by the *win* parameter, which controls where the cursor is placed after the window is refreshed. If the *boolf* parameter is **TRUE**, when the window is refreshed, the cursor is left at the last point where a change was made on the terminal, and the current (*line*, *col*) coordinates for the window specified by the *win* parameter are changed accordingly. If the (*line*, *col*) coordinates are outside the window, the coordinates are forced to (0, 0). If the *boolf* parameter is **FALSE**, when the window is refreshed, the cursor is moved to the current(*line*, *col*) coordinates within the window. The controlling flag is initially set to **FALSE**.

The **leaveok** routine always returns the value **OK**.

char *longname ( )

The **longname** routine returns a pointer to a static area that contains the long (full) name of the terminal as it appears in the **terminfo** entry for the terminal.

mvcur

The **mvcur** routine moves the terminal's cursor from the coordinates specified by the *line* and *col* parameters to the coordinates specified by the *newline* and *newcol* parameters.

It is possible to use this optimization without the benefit of the screen routines. In fact, **mvcur** should not be used with the screen routines. Use **move** and **refresh** to move the cursor position and inform the screen routines of the move.

mvwin (*win*, *line*, *col*)

The **mvwin** routine moves the position of the viewport or the subwindow specified by the *win* parameter from its current starting coordinates to the coordinates specified by the *line* and *col* parameters. The *line* parameter specifies the row on the display for the top row of the window. The *col* parameter specifies the column on the display for the first column of the window.

The **mvwin** routine returns **ERR** if a part of the window position is outside the bounds of the window on which the viewport is defined.

WINDOW *newview (*win*, *numlines*, *numcols*)

The **newview** routine creates a new window that has the number of lines specified by the *numlines* parameter and the number of columns specified by the *numcols* parameter. The new window is a viewport of the window specified by the *win* parameter and starts at the current (*line*, *col*) coordinates of the window specified by the *win* parameter. The resulting window's initial position on the display is set to (0, 0).

The viewport window returned by **newview** is a special subwindow that is suitable for viewport scrolling. Viewport scrolling here refers to the type of scrolling that is characteristic of full–screen editors.

Because the returned viewport window is a subwindow, any change made in either window in the area covered by the viewport window appears in both windows. Both windows actually share the relevant storage area. A viewport window cannot be scrolled using **scroll**.

Other than the exceptions noted above, viewport windows behave like subwindows.

Upon successful completion, a pointer to the control structure for the new viewport is returned.

The **newview** routine returns **ERR** if the window specified by the *win* parameter is a subwindow or a viewport, or if sufficient storage is not available for the new structures.

For Japanese Language Support:

Do not construct a view that includes less than the full line–width of lines containing 2–byte characters.

WINDOW *newwin (*numlines*, *numcols*, *firstline*, *firstcol*)

The **newwin** routine creates a new window that contains the number of lines specified by the *numlines* parameter and the number of columns specified by the *numcols* parameter. The new window will start at the coordinates specified by the *firstline* and the *firstcol* parameters.

If the *numlines* parameter is 0, then that dimension is set to (**LINES** – *firstline*). If the *numcols* parameter is 0, then that dimension is set to (**COLS** – *firstcol*). Therefore, to get a new window of dimensions (**LINES** x **COLS**), use:

```
newwin (0, 0, 0, 0)
```

The size specified for the window can exceed the size of the real display. In this case, a viewport or subwindow must be used to present the data from the window on the terminal.

Upon successful completion, a pointer to the new window structure is returned.

The **newwin** routine returns **ERR** if any of the parameters are invalid, or if there is insufficient storage available for the new structure.

nl ( )
nonl ( )

The **nl** routine sets the terminal to nl mode. When in nl mode, the system maps '\r' (return characters) to '\n' (new–line or line–feed characters). If the mapping is not done, **refresh** can do more optimization. **nonl** turns nl mode off.

The **nl** routine and **nonl** do not affect the way in which **waddch** processes new–line characters.

The **nl** and **nonl** routines always return the value **OK**.

resetty (*boolf*)

The **resetty** routine restores the terminal status flags that were previously saved by **savetty**. If the *boolf* parameter is **TRUE**, then the screen is cleared in addition to resetting the terminal. **resetty** is performed automatically by **endwin** and is not normally called directly by applications.

restore_colors ( )

This routine restores the screen to GREEN foreground color and BLACK background color. This routine is performed automatically by **endwin** if the character variable do_colors is set to **TRUE**.

savetty ( )

The **savetty** routine saves the current terminal status flags. **savetty** is performed automatically by **initscr** and is not normally called directly by applications.

scroll (*win*)

The **scroll** routine moves the data in the window specified by the win parameter up one line and inserts a new blank line at the bottom.

scrollok (*win*, *boolf*)

The **scrollok** routine sets the scroll flag for the window specified by the win parameter. If the *boolf* parameter is **TRUE**, then scrolling is allowed. The default setting is **FALSE**, which prevents scrolling.

setscrreg (*t,b*)
wsetscrreg (*w,t,b*)

The **setscrreg** routine sets the user scrolling region to lines *t* through *b*.

setterm (*name*)
char **name*;

The **setterm** routine sets the terminal characteristics to be those of the terminal specified by the *name* parameter. **setterm** is called by **initscr** so you do not normally have to use it unless you wish to use just the cursor motion optimizations.

WINDOW *subwin (*win*, *numlines*, *numcols*, *firstline*, *firstcol*)

The **subwin** routine creates a subwindow in the window pointed to by the *win* parameter. The subwindow has the number of lines specified by the *numlines* parameter and the number of columns specified by the *numcols* parameter. The new subwindow starts at the coordinates specified by the *firstline* and the *firstcol* parameters. Any change made to the window or the subwindow in the area covered by the subwindow is made to both windows.

The *firstline* and *firstcol* parameters are specified relative to the overall screen, not to the relative (0, 0) of the window specified by the *win* parameter.

If the *numlines* parameter is 0, then the lines dimension is set to (**LINES** – *firstline*). If the *numcols* parameter is 0, then the columns dimension is set to (**COLS** – *firstcol*).

Upon successful completion, a pointer to the control structure for the new subwindow is returned.

The **subwin** routine returns **ERR** if the window specified by the *win* parameter already has a subwindow, or if there is insufficient storage for the new control structure.

touchwin (*win*)

The **touchwin** routine makes it appear as if every location on the window specified by the *win* parameter has been changed. This is useful when overlapping windows are to be refreshed. A subsequent **refresh** request considers all portions of the window as potentially modified. If **touchwin** is not used, then only those positions of the window that have been addressed by an **addch** are inspected.

trackloc (*boolf*)

The **trackloc** routine turns on and off the tracking of the mouse cursor on the screen. If the *boolf* parameter is **TRUE**, then mouse tracking is turned on; if **FALSE**, then it is turned off. By default, mouse tracking is initially turned off.

The keycode **KEY_LOCESC** is returned from **getch** when a mouse report is input. The mouse report is stored in the global **char** array **ESCSTR**, which is 128 bytes long.

Mouse tracking is handled by the **ecpnin** routine. However, the application needs to activate the mouse by using the HFT function call.

Warning: The **meta** and **raw** functions should be performed when the mouse is turned on which will disable all of the control characters (e.g., INTR, Quit, etc.).

tstp ( )

The **tstp** routine saves the current **tty** state and then put the process to sleep. When the process is restarted, the **tty** state is restored and then **wrefresh** (curscr) is called to redraw the screen. The **initscr** routine sets the signal **SIGTSTP** to trap **tstp**.

The **tstp** routine always returns the value **OK**.

#include <cur04.h>
char *unctrl (*xc*)

The **unctrl** routine returns a string that represents the value of the *xc* parameter. Control characters become the lowercase equivalents preceded by a ^ (circumflex). Other letters are unchanged. The **unctrl** routine supports only the characters 0x00 through 0x7F.

Upon successful completion, a pointer to the string for the parameter character is returned.

dounctrl (*boolf*)

The **dounctrl** routine turns the printing of control characters on or off. If the *boolf* parameter is **TRUE**, then the printing is turned on; if **FALSE**, printing is turned off. By default, **dounctrl** processing is initially turned off. The **unctrl** routine defined in **cur04.h** is used to get the string of printable characters being printed. Control characters become the printable character represented by the control character plus 0x40, preceded by a ^ (circumflex).

vscroll (*win*, *numlines*, *numcols*)

The **vscroll** routine scrolls the viewport specified by the *win* parameter on the window.

The *numlines* parameter specifies the direction and amount to scroll up or down. If the *numlines* parameter is positive, the viewport scrolls down the number of lines specified. If the *numlines* parameter is negative, the viewport scrolls up the number of lines specified.

The *numcols* parameter specifies the direction and amount to scroll left or right. If the *numcols* parameter is positive, the viewport scrolls to the right the number of characters specified. If the *numcols* parameter is negative, then the viewport scrolls to the left the number of characters specified.

The **vscroll** routine always scrolls as much of a requested scroll as possible. Specifying a parameter with a magnitude larger than that of the underlying window is not an error.

The **vscroll** routine calls **touchwin** if any scrolling is done.

The **vscroll** routine returns **ERR** if the window specified by the win parameter is not a window created by a call to **newview**.

Display Attributes

Use the following routines to change display attributes. See Changing Display Attributes for information on the external variables that can be changed.

sel_attr (*set*)
int **set*;

The **sel_attr** routine allows you to change the selection and priority of attributes for the run–time terminal. The *set* parameter points to a NULL–terminated integer array that contains display attribute values from the **cur03.h** header file in the order that you want them regardless of whether or not they are available on the terminal.

Groups of attributes (colors and fonts) cannot be split in the array. For instance, all foreground colors specified must be in adjacent locations in the array.

The first element of a group of attributes must be the default color or font of the terminal. For example, the first foreground color specified is usually **F_WHITE**, and the first background color specified is usually **B_BLACK**.

It is recommended that **sel_attr** only be called before **initscr**. If **sel_attr** is called after **initscr**, then the routine **setup_attr** should be called after calling **sel_attr**. If **sel_attr** is called after data has been added to a window, the values in the associated attribute array for that window may denote different attributes than the original attributes used when displaying the data (except **NORMAL**, which remains constant). A subsequent refresh of the window shows the different attributes only if the data has been modified or if a total refresh has been forced by a previous call to **touchwin**.

To use this routine, put the following statement at the beginning of the program file:

```
#include <cur03.h>
```

The **sel_attr** routine always returns the value **OK**.

setup_attr ( )

The **setup_attr** routine creates the display attribute masks assigned to the attribute variables declared in the **cur01.h** header file. The priorities of the attributes determine how the masks are created.

This routine is called by **initscr** and is not normally called by applications. This routine should only be called following a call to **sel_attr**, which follows a call to **initscr**.

Implementation Specifics

The Extended curses Subroutine Library is part of Base Operating System (BOS) Runtime of AIX for RISC System/6000.

Related Information

The **printf** command, **scanf** command, **captoinfo** command.

Curses Programming Example

The following example program, **twinkle.c**, uses the extended curses library routines (**libcur.a**) to create a series of displays on the screen.

```
#include        <cur00.h>
#include <signal.h>

#define NCOLS 80
#define NLINES 24
#define MAXPATTERNS 11

struct locs
{
   char  y, x;
};

typedef struct locs  LOCS;

LOCS  layout[ NCOLS * NLINES ];  /* current board layout */

int   pattern,                   /* current pattern number */
      numstars;                  /* numbers of stars in ptern */

main()
{
   char *getenv();
   int die();

   srand( getpid() );            /* initialize random sequence */
   initscr();
   signal( SIGINT, die );
   noecho();
   leaveok( stdscr, TRUE );
   scrollok( stdscr, FALSE );

   for( ;; )
   {
      makeboard();               /* make the board setup */
      puton( '*' );              /* put on '*'s */
      system( "sleep 2" );
      erase();
      refresh();
   }
}

/*
**  On program exit, move the cursor to the lower left corner by
**  direct addressing, since current location is not certain.
**  We say we used to be at the upper right corner to obtain
**  absolute addressing.
*/

die()
{
   signal( SIGINT, SIG_IGN );
   mvcur( LINES/2, COLS/2, 0, 0 );
```

```
    wclear( curscr );
    wrefresh( curscr );
    endwin();
    exit(0);
}

/*
**  Make the current board setup.  It picks a random pattern and
**  calls ison() to determine if the character is on that pattern
**  or not.
*/

makeboard()
{
    reg int  y, x;
    reg LOCS  *lp;

    pattern = rand() % MAXPATTERNS;
    lp = layout;
    for( y = 0; y < NLINES; y++ )
    {
        for( x = 0; x < NCOLS; x++ )
        {
            if( ison( y, x ))
            {
                lp -> y = y;
                lp++ -> x = x;
            }
        }
    }
    numstars = lp - layout;
}

/*
**  Return TRUE if( y, x ) is on the current pattern.
*/

ison( y, x )

reg int y, x;

{
    switch( pattern )
    {
        /*
        ** Alternating lines:
        */
        case 0:
            return !( y & 01 );
        /*
        ** Box:
        */
        case 1:
            if( y < 3 || y >= NLINES - 3 )
                return TRUE;
            return( x < 4 || x >= NCOLS - 4 );
        /*
        ** Cross:
        */
```

```
case 2:
   return( ( x + y ) & 01 );
/*
** Bar across center:
*/
case 3:
   return( y >= 9 && y <= 15 );
/*
** Alternating columns:
*/
case 4:
   return !( x & 02 );
/*
** Bar down center:
*/
case 5:
   return( x >= 36 && x <= 44 );
/*
** Bar across and down center:
*/
case 6:
  return( ( y >= 9 && y <= 15 ) || ( x >= 37 && x <= 43 ));
/*
** Bar across and down center, in a box:
*/
case 7:
   if( y < 3 || y >= NLINES - 3 )
      return TRUE;
   if( x < 4 || x >= NCOLS - 4 )
      return TRUE;
   return( ( y >= 10 && y <= 14 )||( x >= 36 && x <= 44 ));
/*
** Asterisk:
*/
case 8:
   if( abs( x - y ) <= 2 || abs( NLINES - ( x + y )) <= 2 )
      return TRUE;
   if( abs( ( NLINES/2 ) - x ) <= 2 )
      return TRUE;
   return( abs( ( NLINES/2 ) - y ) <= 1 && x <= NLINES );
/*
**  Ellipse:
*/
case 9:
   return
   (
      (
        (( float ) (( x-40 ) * ( x-40 )) ) / 1521 +
        (( float ) (( y-12 ) * ( y-12 )) ) / 121
      ) <= 1
   );
/*
** Circle:
*/
case 10:
   return
      (
         (
           (( float ) (( x-28 ) * ( x-28 )) ) / 729 +
```

```
                (( float ) (( y-12 ) * ( y-12 )) ) / 121
            ) <= 1
        );
    }    /* end of switch( pattern ) */
}        /* not reached */

puton(ch)
reg char ch;
{
    reg LOCS  *lp;
    reg LOCS  *end;
    LOCS      temp;
    reg  int     r;

    end = &layout[ numstars ];
    for( lp = layout; lp < end; lp++ )
    {
        r = rand() % numstars;
        temp = *lp;
        *lp = layout[ r ];
        layout[ r ] = temp;
    }

    for( lp = layout; lp < end; lp++ )
    {
        mvaddch( lp -> y, lp -> x, ch );
        refresh();
    }
}  /* end of twinkle */
```

Appendix A. Enhanced X-Windows Xlib Data Structures

The **XConfigureRequestEvent** data structure
The **XMapRequestEvent** data structure
The **XResizeRequestEvent** data structure
The **XColormapEvent** data structure
The **XClientMessageEvent** data structure
The **XPropertyEvent** data structure
The **XSelectionClearEvent** data structure
The **XSelectionRequestEvent** data structure
The **XSelectionEvent** data structure
The **XErrorEvent** data structure
The **XWMHints** data structure
The **XSizeHints** data structure
The **XIconSize** data structure
The **XClassHint** data structure
The **XrmValue** data structure
The **XrmOptionDescList** data structure

XVisualInfo Data Structure

```
#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

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;
```

The fields of the **XVisualInfo** data structure are as follows:

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

blue_mask — Defined only for **DirectColor** and **TrueColor**. Each mask has one contiguous set of bits with no intersections.

class — 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 video 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 other hardware. The visual types affect the colormap and the RGB values in the following ways:

PseudoColor — A pixel value indexes a colormap to produce independent RGB values, and the RGB values can be changed dynamically.

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.
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.
TrueColor	A pixel value is decomposed into separate RGB subfields, 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.
StaticColor	A pixel value indexes a colormap to produce independent RGB values, and the RGB values can be changed dynamically, except that the colormap has predefined read-only server-dependent RGB values.
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 that results in shades of gray. **StaticGray** with a two-entry colormap can be considered monochrome.

colormap_size	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.
depth	Specifies the depth of the screen.
green_mask	Defined only for **DirectColor** and **TrueColor**. Each mask has one contiguous set of bits with no intersections.
red_mask	Defined only for **DirectColor** and **TrueColor**. Each mask has one contiguous set of bits with no intersections.
screen	Specifies the screen.
visual	Specifies the visual.
visualid	Specifies the visual ID.

Related Information

The **XGetVisualInfo** subroutine, **XMatchVisualInfo** subroutine, **XVisualIDFromVisual** subroutine

XSetWindowAttributes Data Structure

```
#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)

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

The following table lists the defaults for each window field and indicates if the field is applicable to **InputOutput** or **InputOnly** windows.

Window Field	Default Values	InputOutput	InputOnly
background_pixmap	**None**	Yes	No
background_pixel	Undefined	Yes	No
border_pixmap	**CopyFromParent**	Yes	No
border_pixel	Undefined	Yes	No
bit_gravity	**ForgetGravity**	Yes	No
win_gravity	**NorthWestGravity**	Yes	Yes
backing_store	**NotUseful**	Yes	No
backing_planes	All 1s	Yes	No
backing_pixel	**0**	Yes	No

save_under	**False**	Yes	No
event_mask	empty set	Yes	Yes
do_not_propagate_mask	empty set	Yes	Yes
override_redirect	**False**	Yes	Yes
colormap	**CopyFromParent**	Yes	No
cursor	**None**	Yes	Yes

The fields of the **XSetWindowAttributes** data structure are as follows:

background_pixmap Specifies the pixmap to be used for a window background. This pixmap can be any size, but some sizes are faster than others. Use the **XQueryBest Sizes** subroutine to determine the optimum size. The *background_pixmap* field can be set to a pixmap ID, or either the value of **None** or **ParentRelative**. The default is the value of **None**. The *background_pixmap* field and the window must have the same depth, or a **BadMatch** error is returned.

Only **InputOutput** windows can have backgrounds.

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* field is not the value of **None**.

- If the *background_pixmap* field is set to the value of **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. **Expose** events are then generated for the regions, even if the *background_pixmap* field is the value of **None**.
- If the *background_pixmap* field is set to **ParentRelative**, the following occurs:
  - The *background_pixmap* field of the parent window is used if the child window has the same depth as the parent window, or a **BadMatch** error is returned.
  - The window has no defined background. If the parent window has a *background_pixmap* field of the value of **None**, the window also has a *background_pixmap* field of the value of **None**.
  - A copy of the *background_pixmap* field of the parent window is not made. The *background_pixmap* field of the parent window is examined each time the *background_pixmap* field 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 the *background_pixmap* field overrides any previous *background_pixmap* field. The *background_pixmap* field 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 Specifies a pixel value of a single color for the background of the window. This field can be set to any pixel value. The default value for the *background_pixel* field is undefined.

If the *background_pixel* field is specified, it overrides the default *background_pixmap* field or any value set in the *background_pixmap* field, and a pixmap of undefined size is created and filled with the specified pixel and used for the background. All pixels, in the background of the window, will be set to this value. Range checking is not performed on the pixel, as it is truncated to the appropriate number of bits.

Setting a new background with the *background_pixel* field overrides any previous background.

border_pixmap Specifies the pixmap for the border of a window. This pixmap can be any size. The *border_pixmap* field and the child window must have the same depth, or a **BadMatch** error is returned.

Only **InputOutput** windows can have a border.

Setting a new border with the *border_pixmap* field overrides any previous border. Setting a *border_pixmap* field value overrides the default value. The default value is **CopyFromParent**.

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

The pixmap used for the *border_pixmap* field can be freed immediately if no further explicit reference to it is made. If the pixmap used for the *border_pixmap* field is freed, the X Server may or may not keep a copy of it. The X Server can use the same pixmap each time the window is repainted or it may make a copy.

border_pixel Specifies a pixel value to be used for the window border. The server creates a pixmap of unspecified size filled with the pixel for the window border. The border tile origin is always the same as the background tile origin. The default for the *border_pixel* field is undefined. Range checking is not performed on the pixel, as it is truncated to the appropriate number of bits.

Only **InputOutput** windows can have a border.

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

The output to a window is always clipped to the inside of the window so that graphics operations are not affected by the border.

bit_gravity Specifies which region of the window should be retained when an **InputOutput** window is resized. The default *bit_gravity* is the

ForgetGravity value. 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* field of the window. The values for the *bit_gravity* field include:

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 **Expose** events are generated. If no background is defined, the existing screen contents are not altered. Some X Servers may ignore the specified *bit_gravity* field and always generate exposure events.

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), the change in position of each pixel becomes (-x, -y).

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

If the inside width or height of a window is not changed and if the window is moved or its border is changed, 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)
SouthEastGravity	(Width, Height)

When a window with one of these *bit_gravity* field values is resized, the corresponding pair defines the change in position of each pixel in the window.

win_gravity Specifies how the **InputOutput** or **InputOnly** window should be repositioned if the parent window is resized. Changing the inside width

or height of the window causes child windows to be reconfigured, depending on the specified *win_gravity* field. The default for the *win_gravity* field is the value of **NorthWestGravity**.

If the inside width or height of a window is not changed and if the window is moved or its border is changed, 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)
SouthEastGravity	(Width, Height)

When a window with one of these *win_gravity* field values has its parent window resized, the corresponding pair defines the change in position of the window within the parent window. When the window is repositioned, a **GravityNotify** event is generated.

StaticGravity If the change in size of the window is coupled with a change in position (x, y), the change in position of a child window when the parent window is resized becomes (-x, -y).

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

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

backing_store Advises the X Server what to do with the contents of a window. Some implementations may choose to maintain the contents of **InputOutput** windows. If the X Server maintains the contents of a window, the pixels saved offscreen are known as the backing store. This field can be set to the following values:

NotUseful Advises the X Server that maintaining contents is not necessary. Some X implementations can still maintain contents; therefore, exposure events are not generated. This is the default value.

WhenMapped Advises the X Server that maintaining contents of obscured regions when the window is mapped would be beneficial. The X Server can generate an **Expose** event when the window is created.

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. While the X Server maintains the contents of the window, **Expose** events are not normally generated. The X Server can stop maintaining contents at any time.

When the contents of obscured regions of a window are being maintained, the regions obscured by noninferior windows are included in the destination of graphics requests (and source, when the window is the source). Regions obscured by inferior windows, however, are not included.

backing_planes Indicates (with one bits) which bit planes of the **InputOutput** window hold dynamic data that must be preserved in the backing store and during save unders. If you request backing store or save unders, the *backing_planes* field will minimize the amount of off–screen memory required to store your window. The default is all bits set to the value of **1**.

backing_pixel Specifies the values to use in planes not covered by the *backing_planes* field. The X Server is free to save only the specified bit planes in the backing store or the save under and is free to regenerate the remaining planes with the specified pixel value. Any extraneous bits in these values, beyond the depth of the window, can be ignored. If you request backing store or save unders, the *backing_pixel* field will minimize the amount of off–screen memory required to store your window. The default is the value of **0**.

save_under If the *save_under* field is the value of **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 is the value of **False**.

Some server implementations can preserve bits of **InputOutput** windows under other **InputOutput** 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 Defines events the client is interested in for this **InputOutput** or **InputOnly** window or, in some cases, for the inferiors of the window. This mask is the bitwise-inclusive OR of one or more of the valid event mask bits. If the **NoEventMask** constant is specified, no maskable events are reported. The default is the empty set.

do_not_propagate_mask

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 of one or more of the valid event mask bits. If the **NoEventMask** constant is specified, no maskable events are reported. The default is the empty set.

override_redirect

Specifies if a map or configure request on a window should override a **SubstructureRedirectMask** request on the parent window. The default is the value of **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, the *override_redirect* field must be used.

colormap

Specifies the colormap, if any, that best reflects the true colors of an **InputOutput** window. The colormap must have the same visual type as the window, or a **BadMatch** error is returned. The *colormap* field can be set to a specific colormap or to the value of **CopyFromParent**, which is the default.

If the *colormap* field is set to the value of **CopyFromParent**, the colormap of the parent window is copied and used by the child window. However, the child window must have the same visual type as the parent, or a **BadMatch** error is returned. A **BadMatch** error is also returned if the parent window has the *colormap* field is set to the value of **None**. The colormap is copied by sharing the colormap object between the child and parent windows, not by making a complete copy of the colormap contents. Subsequent changes to the parent window do not affect the child window.

cursor

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

If the value of **None** is specified, the cursor of the parent is used when the pointer is in the **InputOutput** or **InputOnly** window. Any change in the parent cursor will change the displayed cursor immediately. Use the **XFreeCursor** subroutine to free the cursor immediately if no further explicit reference to it is made.

XWindowChanges Data Structure

```
#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)


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

The fields of the **XWindowChanges** data structure are as follows:

x — Specifies the position of the x coordinate of the upper-left outer corner of the window. This coordinate is relative to the origin of the parent window.

y — Specifies the position of the y coordinate of the upper-left outer corner of the window. This coordinate is relative to the origin of the parent window.

width — Specifies the width of the inside of the window, excluding the border. This field should be a nonzero value or a **BadValue** error is returned. Attempts to configure a root window have no effect.

height — Specifies the width of the inside of the window, excluding the border. This field should be a nonzero value or a **BadValue** error is returned. Attempts to configure a root window have no effect.

border_width — Specifies the width of the border in pixels. Changing only the *border_width* field leaves the outer-left corner of the window in a fixed position, but moves the absolute position of the window origin. Attempts to change the *border_width* field on an **InputOnly** window will result in a **BadMatch** error.

sibling — Specifies the sibling window for stacking operations. If the *sibling* field is specified without the *stack_mode* field, a **BadMatch** error will result.

stack_mode — Specifies how the window is to be restacked. The *stack_mode* field can be the **Above**, **Below**, **TopIf**, **BottomIf**, or **Opposite** value.

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

StackMode	**Sibling Specified.**
Above	The window is placed just above the sibling window.

Below	The window is placed just below the sibling window.
TopIf	If the sibling window occludes the window, the window is placed at the top of the stack.
BottomIf	If the window occludes the sibling window, the window is placed at the bottom of the stack.
Opposite	If the sibling window occludes the window, the window is placed at the top of the stack. Otherwise, if the window occludes the sibling window, the window is placed at the bottom of the stack.

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

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

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

Otherwise, if some other client has selected the **ResizeRedirectMask** value 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* field of the window does not affect the **ResizeRedirectMask** value, and the **SubstructureRedirectMask** value on the parent window has precedence over the **ResizeRedirectMask** value on the window.

When the geometry of the window is changed as specified, the window is restacked among sibling windows, and a **ConfigureNotify** event is generated if the state of the window actually changes. The X Server generates **GravityNotify** events after generating **ConfigureNotify** events. If the inside width or height of the window has changed, the children of the window are affected as follows:

- If the size of the window actually changes, the subwindow may move according to the window gravity. Depending on the window's bit gravity, the contents of the window may also be moved.
- If regions of the window were obscured but are not now, exposure processing is performed on formerly obscured windows, including the window itself and its inferiors.
- By increasing the width or height, exposure processing is also performed on any new regions of the window and on any regions where window contents are lost.
- The restack check, specifically the computation for the **BottomIf**, **TopIf**, and **Opposite** values, is performed with respect to the final size and position of the window as controlled by the other fields of the request, not the initial position of the window. A *sibling* field should be specified with a *stack_mode* field.

Related Information

The **XChangeWindowAttributes** subroutine, **XCirculateSubwindows** subroutine, **XCirculateSubwindowsDown** subroutine, **XCirculateSubwindowsUp** subroutine, **XConfigureWindow** subroutine, **XLowerWindow** subroutine, **XMoveResizeWindow** subroutine, **XMoveWindow** subroutine, **XRaiseWindow** subroutine, **XResizeWindow** subroutine, **XRestackWindows** subroutine, **XSetWindowBackground** subroutine, **XSetWindowBackgroundPixmap** subroutine, **XSetWindowBorder** subroutine, **XSetWindowBorderPixmap** subroutine, **XSetWindowBorderWidth** subroutine, **XTranslateCoordinates** subroutine.

The **ChangeWindowAttributes** protocol request, **CirculateWindow** protocol request, **ConfigureWindow** protocol request.

XWindowsAttributes Data Structure

```
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 window_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 do_not_propagate_mask;  /* set of events that should not
                                    propagate */
    Bool override_redirect;     /* boolean value for
                                    override-redirect */
    Screen *screen;             /* back pointer to correct screen */
} XWindowAttributes;
```

The fields of the **XWindowAttributes** data structure are as follows:

x — Defines the x coordinate of the drawable. If the drawable is a window, this coordinate specifies the upper-left outer-corner of the window relative to the origin of the parent window. If the drawable is a pixmap, this coordinate is set to a value of 0.

y — Defines the x coordinate of the drawable. If the drawable is a window, this coordinate specifies the upper-left outer-corner of the window relative to the origin of the parent window. If the drawable is a pixmap, this coordinate is set to the value of **0**.

width — Specifies the width of the inside of the window, excluding the border, for a window.

height — Specifies the height of the inside of the window, excluding the border, for a window.

border_width — Specifies the width of the border in pixels. If the drawable is a pixmap, this field is set to the value of **0**.

depth — Specifies the depth of the pixmap or window in bits per pixel for the object. The depth must be supported by the root window of the specified drawable.

visual — Specifies a pointer to the **Visual** structure associated with the screen.

root — Specifies the root ID of the screen containing the window.

class — Specifies the window class, which is either the **InputOutput** or **InputOnly** value.

bit_gravity — Specifies the bit gravity of the window. The *bit_gravity* field can be set to one of the following values:

CenterGravity **SouthGravity**

EastGravity **SouthEastGravity**

ForgetGravity **SouthWestGravity**

NorthGravity **StaticGravity**

NorthEastGravity **WestGravity**

NorthWestGravity

win_gravity — Specifies the gravity of the window. The *win_gravity* field can be set to one of the following values:

CenterGravity **SouthGravity**

EastGravity **SouthEastGravity**

ForgetGravity **SouthWestGravity**

NorthGravity **StaticGravity**

NorthEastGravity **WestGravity**

NorthWestGravity

backing_store — Indicates how the X Server should maintain the contents of a window. It can be set to the **NotUseful**, **WhenMapped**, or **Always** value.

backing_planes — Indicates which bit planes of the window that hold dynamic data must be preserved in *backing_store* and during *save_under*.

backing_pixel — Indicates the values to use when restoring planes from a partial backing store.

save_under — Indicates if the area under the newly mapped windows should be saved and restored. It can be either the value of **True** or **False**.

colormap — Indicates colormap for the window specified. It can be set to a colormap ID or to the value of **None**.

map_installed — Indicates if the colormap is currently installed. It can be either the value of **True** or **False**.

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

all_event_masks Specifies the bitwise inclusive-OR of all event masks selected on the window by interested clients.

your_event_mask Specifies the bitwise inclusive-OR of all event masks selected by the querying client.

do_not_propagate_mask
Specifies the bitwise inclusive-OR gate or operation of the set of events that should not propagate.

override_redirect Indicates if this window overrides structure control facilities. It can be either the value of **True** or **False**. If the *override_redirect* field is the value of **True**, window manager clients usually should ignore the window.

Transient windows should mark the windows associated with them.

screen Returns a pointer to the correct screen.

Related Information

The **XGetGeometry** subroutine, **XGetWindowAttributes** subroutine, **XQueryPointer** subroutine, **XQueryTree** subroutine.

XColor Data Structure

```
typedef struct {
    unsigned long pixel;                /* pixel value */
   unsigned short red, green, blue;    /* RGB values */
    char flags;                         /* DoRed, DoGreen, DoBlue */
   char pad;
} XColor;
```

The fields of the **XColor** data structure are as follows:

blue — Specifies a scale between 0 and 65535. That is, 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.

flags — Specifies which colors are to be set. The *flags* field can be one or more of the **DoRed**, **DoGreen**, or **DoBlue** values.

green — Specifies a scale between 0 and 65535. That is, 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.

pad — Indicates to fill unused positions in a field with dummy data, usually zeros or blanks.

pixel — Specifies the value of the pixel.

red — Specifies a scale between 0 and 65535. That is, 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.

Related Information

The **XCopyColormapAndFree** subroutine, **XCreateColormap** subroutine, **XFreeColormap** subroutine, **XSetWindowColormap** subroutine.

The **ChangeWindowAttributes** protocol request, **CopyColormapAndFree**, **CreateColormap** protocol request, **FreeColormap** protocol request.

XGCValues Data Structure

```
#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)


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, or
                                 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 or 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.

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

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* field.
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* field.
CapProjecting	thin	A single pixel is drawn.

For a line with concident endpoints (x1=x2, y1=y2), when the *join_style* field 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* field is applied at both endpoints.

The fields of the **XGCValues** data structure are as follows:

function — Specifies the logical operation to be performed.

plane_mask — Specifies the operations to a subset of planes that need to be restricted. The result is computed by the following:

```
((src FUNC dst) AND plane-mask) OR (dst AND (NOT pla
ne-mask))
```

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

line_width — Specifies the width of the line, 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 graphics request.

Unless otherwise specified by the *join_style* or the *cap_style* fields, the bounding box of a wide line with endpoints [x1, y1] to [x2, y2] and 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
)],
[x1-(w*sn/2), y1+(w*cs/2)], [x1+(w*sn/2), y1-(w*cs/2
)]
```

In this example, 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 line and is drawn that way only if the center of the pixel is fully inside the bounding box, which has infinitely thin edges. If the center of the pixel is exactly on the bounding box, it is part of the line only if the interior is immediately to its right (the x increasing direction). Pixels with centers on a horizontal edge are a special case and are part of the line only if the interior or the boundary is immediately below (the y increasing direction) and the interior or the boundary is immediately to the right (the x increasing direction).

Thin lines (zero line width) are one pixel wide lines drawn using an unspecified, device-dependent algorithm. There are only the following two constraints on this algorithm:

- If a line is drawn unclipped from [x1, y1] to [x2, y2] and if another line is drawn unclipped from [x1+dx, y1+dy] to [x2+dx, y2+dy], a point [x, y] is touched by drawing the first line only if the point [x+dx, y+dy] is touched by drawing the second line.
- The effective set of points comprising a line cannot be affected by clipping. That is, a point is touched in a clipped line 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 the *cap_style* and the *join_style* fields. A *line_width* field of 0 may differ from a *line_width* field 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.

line_style

Defines which sections of a line are drawn. The *line_style* field can be one of the following values:

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 the **CapButt** value used where even and odd dashes meet.

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

cap_style Defines how the endpoints of a path are drawn. The *cap_style* field can be one of the following values:

CapNotLast Equivalent to the **CapButt** value, except that for a *line_width* field 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* field, centered on the endpoint (equivalent to the **CapButt** value for a *line_width* field 0 or 1).

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

join_style Defines how corners are drawn for wide lines. The *join_style* field can be one of the following values:

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

JoinRound A circular arc with diameter equal to the *line_width* field, centered on the join point.

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

tile Specifies the tile to be used. The tile pixmap must have the same root window and depth as the graphics context, or a **BadMatch** error results.. The *tile* field is interpreted relative to the origin of whatever destination drawable is specified in a graphics request.

stipple Specifies the stipple to be used. The stipple pixmap must have depth one and must have the same root window as the GC, or a **BadMatch** error results. For stipple operations where the *fill_style* field 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* field. Any size pixmap can be used for tiling or stippling although some sizes may be faster to use than others.The *stipple* field is interpreted relative to the origin of whatever destination drawable is specified in a graphics request.

clip_x_origin Specifies that the field is interpreted relative to the origin of whatever destination drawable is specified in a graphics request.

clip_y_origin

Specifies that the field is interpreted relative to the origin of whatever destination drawable is specified in a graphics request.

fill_style

Specifies that the field defines the contents of the source for line, text, and fill requests. For all text and fill requests, for line requests with *line_style* field of **LineSolid**, and for the even dashes for line requests with *line_style* field of **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 the *line_style* field value of **LineDoubleDash**, the odd dashes are controlled by the *fill_style* field in the following manner:

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

dashes

Specifies the dash list for the dashed line style to be set for the specified **GC**.

The value allowed by the *dashes* field is a simplified form of the more general patterns that can be set with the **XSetDashes** subroutine. Specifying a value of `N` is equivalent to specifying the two-element list `[N, N]` in the **XSetDashes** subroutine. `N` specifies the length of the dash list. This value must be nonzero, or a **BadValue** error results. The default dash list in a newly created **GC** is equivalent to [4,4].

dash_offset

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.

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 horizointal 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 value for the dash list in a newly created **GC** is equivalent to [4, 4].

clip_mask Restricts writes to the destination drawable. If a pixmap is specified as the *clip_mask* field, it must have a depth of one and have the same root window as the **GC**. If the *clip_mask* field is the value of **None**, the pixels are always drawn, regardless of the clip origin. The *clip_mask* field can also be set with the **XSetClipRectangles** or **XSetRegion** subroutines. Only pixels where the *clip_mask* field has a 1-bit are drawn. Pixels are not drawn outside the area covered by the *clip_mask* field or where the *clip_mask* field has a 0 bit. It affects all graphics requests.The *clip_mask* field does not clip sources.The *clip_mask* field origin is interpreted relative to the origin of the specified destination drawable in the graphics request.

subwindow_mode Specifies how the subwindow should be clipped. The *subwindow_mode* field can be one of the following values:

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

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

fill_rule Defines which pixels are inside (drawn) for paths given in the **XFillPolygon** subroutine. The *fill_rule* field can be set to one of the following values:

EvenOddRule Specifies that a point is inside if an infinite ray with the point as origin crosses the path an odd number of times.

WindingRule Specifies that 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.

arc_mode Controls filling in the **XFillArcs** subroutine. The *arc_mode* field can be one of the following values:

ArcPieSlice Specifies that arcs are pie-sliced filled.

ArcChord Specifies that arcs are chord-filled.

graphics_exposures Controls the **GraphicsExpose** event generation for the **XCopyArea** and **XCopyPlane** subroutines and any similar requests defined by extension subroutines. The **GraphicsExpose** events are sent even when they are not explicitly requested. To suppress them, set the *graphics_exposures* field to the value of **False**.

Related Information

The **AllPlanes** macro.

The **XChangeGC** subroutine, **XCopyGC** subroutine, **XCreateGC** subroutine, **XFreeGC** subroutine, **XQueryBestSize** subroutine, **XQueryBestStipple** subroutine, **XQueryBestTile** subroutine, **XSetArcMode** subroutine, **XSetBackground** subroutine, **XSetClipMask** subroutine, **XSetClipOrigin** subroutine, **XSetClipRectangles** subroutine, **XSetDashes** subroutine, **XSetFillRule** subroutine, **XSetFillStyle** subroutine, **XSetFont** subroutine, **XSetForeground** subroutine, **XSetFunction** subroutine, **XSetGraphicsExposures** subroutine, **XSetLineAttributes** subroutine, **XSetPlaneMask** subroutine, **XSetState** subroutine, **XSetStipple** subroutine, **XSetSubwindowMode** subroutine, **XSetTile** subroutine, **XSetTSOrigin** subroutine

XStandardColormap Data Structure

```
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 fields in the **XStandardColormap** data structure include the following:

colormap — Specifies the ID of a colormap created by the **XCreateColormap** subroutine.

red_max — Specifies the maximum red values.

green_max — Specifies the maximum green values.

blue_max — Specifies 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.

red_mult — Specifies the scale factors used to compose a full pixel value.

green_mult — Specifies the scale factors used to compose a full pixel value.

blue_mult — Specifies 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.

base_pixel — Specifies the base pixel value used to compose a full pixel value. The *base_pixel* field value is usually obtained from the **XAllocColorPlanes** subroutine.

Given integer red, green, and blue coefficients 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 **GrayScale** 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
```

XSegment Data Structure

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

All *x* and *y* fields are 16-bit signed 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.

Related Information

The **XDrawLine** subroutine, **XDrawLines** subroutine, **XDrawRectangle** subroutine, **XDrawRectangles** subroutine, **XDrawSegments** subroutine.

The **PolySegment** protocol request.

XRectangle Data Structure

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

All *x* and *y* fields are 16-bit signed integers. The *width* and *height* fields 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.

Related Information

The **XDrawLine** subroutine, **XDrawLines** subroutine, **XDrawRectangle** subroutine, **XDrawRectangles** subroutine, **XDrawSegments** subroutine.

The **PolyRectangle** protocol request

XPoint Data Structure

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

All *x* and *y* fields 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.

Related Information

The **XDrawPoint** subroutine, **XDrawPoints** subroutine.

XArc Data Structure

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

All *x* and *y* fields are 16-bit signed integers. The *width* and *height* fields 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.

Related Information

The **XDrawArc** subroutine, **XDrawArcs** subroutine.

XCharStruct Data Structure

```
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
                                   original */
    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;
```

The **XCharStruct** data structure defines the bounding box of a single character, a string, or the overall characteristics of a font. A nonexistent character is represented with all fields of the **XCharStruct** data structure set to the value of **0**. Any of the fields of the **XCharStruct** data structure can be negative.

The fields of the **XCharStruct** data structure are defined as follows:

lbearing Specifies the extent of the left edge of the character ink from the origin.

rbearing Specifies the extent of the right edge of the character ink from the origin.

width Specifies the logical width of the character. If the *width* field is negative, the next character is placed to the left of the current origin.

ascent Specifies the extent of the top edge of the character ink from the origin.

descent Specifies the extent of the bottom edge of the character ink from the origin.

If the baseline is at the y coordinate *y*, the logical extent of the font is inclusive between the y coordinate values (`y - font.ascent`) and (`y + font.descent - 1`). Typically, the minimum interline spacing between rows of text is given by ascent + descent.

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 + width, y]
```

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

When `lbearing` is the value of **0**, no pixels with the X-coordinate less than *x* are drawn. Any of these values can be negative.

attributes Specifies the per character flags. The X protocol does not define the interpretation of the *attributes* field.

The baseline (the y position of the character origin) is logically viewed as being the scan line just below nondescending characters. When *descent* is 0, only pixels with Y coordinates less than y are drawn, and the origin is logically viewed as being coincident with the left edge of a nonkerned character. When *lbearing* is zero, no pixels with X coordinates less than x are drawn. Any of the **XCharStruct** metric members could be negative. If the *width* is negative, the next character will be placed to the left of the current origin. The X protocol does not define the interpretation of the *attributes* member in the **XCharStruct** structures. A nonexistent character is represented with all members of its **XCharStruct** structure set to zero.

XFontProp Data Structure

```
typedef struct {
    Atom name;
   unsigned long card32;
} XFontProp;
```

The **XFontProp** data structure describes a font property. A pointer to a list of these properties is included in the **XFontStruct** data structure.

XChar2b Data Structure

```
typedef struct {        /* normal 16-bit characters are 2 bytes */
   unsigned char byte1;
   unsigned char byte2;
} XChar2b;
```

The **XChar2b** data structure is used in the **Xlib** library subroutines that use 2-byte matrix fonts.

Enhanced 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.

XFontStruct Data Structure

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

```
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
                                    nonzero 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 fields of the **XFontStruct** data structure include the following:

direction Provides a hint as to what most **XCharStruct** elements have in terms of character-width metric. The Core protocol does not support vertical text. The *direction* field can be set to one of the following:

FontLeftToRight Specifies a positive character-width metric.

FontRightToLeft Specifies a negative character-width metric.

min_byte1 Indicates the first row that exists.

max_byte1 Indicates the last row that exists.

min_char_or_byte2 Specifies the linear character index of the last element. If the *min_byte1* and *max_byte1* fields are both the value of **0**, then the *max_char_or_byte2* field specifies the linear character index corresponding to the first element of the *per_char* field array. If either the *min_byte1* or *max_byte1* field is nonzero, then both the

min_char_or_byte2 and *max_char_or_byte2* fields are less than 256, and the two-byte character index values corresponding to the *per_char* field 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
```

per_char — Specifies information about each character. If the *per_char* field is the value of **NULL**, all glyphs between the first and last character, indexes inclusive, have the same information as given by both the *min_bounds* and *max_bounds* fields.

all_chars_exist — If the *all_chars_exist* field is the value of **True**, all characters in the *per_char* field array have nonzero bounding boxes.

default_char — Specifies the character to use when an undefined or nonexistent character is specified by the client. 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.

min_bounds — Specifies the smallest rectangle enclosing the shape obtained by superimposing all of the characters at the same origin [x,y].

max_bounds — Specifies 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 baseline coordinates 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
```

ascent — Specifies the logical extent of the font above the baseline that is used for determining line spacing. Specific characters can extend beyond this.

descent — Specifies the logical extent of the font at or below the baseline that is used for determining line spacing. Specific characters may extend beyond this.

The interpretation of the *attributes* field in the **XCharStruct** data structure is not defined by the core protocol. A nonexistent character is represented with members of the **XCharStruct** data structure set to the value of **0**.

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, specified in pixels.
NORM_SPACE	unsigned	The normal interword spacing, specified in pixels.
MAX_SPACE	unsigned	The maximum interword spacing, specified in pixels.
END_SPACE	unsigned	The additional spacing at the end of sentences, specified in pixels.
SUPERSCRIPT_X SUPERSCRIPT_Y	integers	The offset, specified in pixels from character origin where superscripts should begin. If the origin is at `[x,y]`, then superscripts should begin at `[x + SUPERSCRIPT_X, y – SUPERSCRIPT_Y]`.
SUBSCRIPT_X SUBSCRIPT_Y	integers	Offset, specified 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	The y offset, specified in pixels from the baseline to the top of an underline. If the baseline is the y coordinate *y*, then the top of the underline is at `(y + UNDERLINE_POSITION)`.
UNDERLINE_THICKNESS	unsigned	Thickness of the underline, specified in pixels.
STRIKEOUT_ASCENT STRIKEOUT_DESCENT	integers	Vertical extents, specified 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	The y offset, specified in pixels from the baseline to the top of the capital letters, ignoring accents. If the baseline is at the y coordinate *y*, then the top of the uppercase letters is at (`y - CAP_HEIGHT`).
WEIGHT	unsigned	The weight or boldness of the 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.

Related Information

The **XFreeFont** subroutine, **XFreeFontInfo** subroutine, **XFreeFontNames** subroutine, **XFreeFontPath** subroutine, **XGContextFromGC** subroutine, **XGetFontPath** subroutine, **XGetFontProperty** subroutine, **XListFonts** subroutine, **XListFontsWithInfo** subroutine, **XLoadFont** subroutine, **XLoadQueryFont** subroutine, **XQueryTextExtents** subroutine, **XQueryTextExtents16** subroutine, **XSetFontPath** subroutine, **XTextExtents** subroutine, **XTextExtents16** subroutine, **XTextWidth** subroutine, **XTextWidth16** subroutine, **XUnloadFont** subroutine.

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;
```

If the *font* field is the value of **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.

Related Information

The **XDrawText** subroutine

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;
```

The fields of the **XTextItem16** data structure are as follows:

chars Specifies a pointer to two-byte characters. The *chars* field 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* field of the **XChar2b** structure is taken as the most-significant byte.

nchars Specifies the number of characters.

delta Specifies the delta between strings along the x axis.

font Specifies the font to print it in. If the *font* field is the value of **None**, the font is changed before printing and stored in the **GC**. If an error is generated during text drawing, the font in the **GC** is undefined.

Related Information

The **XDrawText16** subroutine

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;            /* quant. of scanline 8, 16, 32 */
    int bitmap_bit_order;       /* MSBFirst or LSBFirst */
    int bitmap_pad;             /* 8, 16, 32 either 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 green_mask;   /* bits in z arrangement */
    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 fields in this data structure, for example, *height*, *width*, and *xoffset*. These changes create a subset of the image. Other fields of this structure, for example, byte_order and bitmap_unit, are characteristic of both the image and the server. If these fields 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)).
```

Related Information

The **XAddPixel** subroutine, **XCreateImage** subroutine, **XDestroyImage** subroutine, **XGetImage** subroutine, **XGetPixel** subroutine, **XGetSubImage** subroutine, **XPutImage** subroutine, **XPutPixel** subroutine, **XSubImage** subroutine.

XKeyboardControl Data Structure

```
#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)


typedef struct {
   int key_click_percent;
   int bell_percent;
   int bell_pitch;
   int bell_duration;
   int led;
   int led_mode;          /* LedModeOn, LedModeOff */
   int key;
   int auto_repeat_mode;  /* AutoRepeatModeOff, AutoRepeatModeOn,
                             AutoRepeatModeDefault */
} XKeyboardControl;
```

The fields of the **XKeyboardControl** data structure include the following values:

key_click_percent — 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.

bell_percent — 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.

bell_pitch — 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.

bell_duration — Sets the duration of the bell (in milliseconds), if possible. A setting of -1 restores the default. Other negative values generate an error.

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

led_mode — 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*, a **BadMatch** error is generated. No standard interpretation of LEDs is defined.

key — Specifies a key on the keyboard.

auto_repeat_mode Specifies the auto repeat mode. If the *auto_repeat_mode* and the *key* fields are specified, the *auto_repeat_mode* of that key is changed, if possible. If only the *auto_repeat_mode* field 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 the *auto_repeat_mode* field, a **BadMatch** 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.

XKeyboardState Data Structure

```
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 the *led_mask* field corresponds to LED 1, and each bit in *led_mask* that is set to 1 indicates an LED that is lit.

The *auto_repeats* field 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* field can be set to the value of **AutoRepeatModeOn** or **AutoRepeatModeOff**.

Related Information

The **XAutoRepeatOn** subroutine, **XBell** subroutine, **XChangeKeyboardControl** subroutine, **XGetKeyboardControl** subroutine, **XGetPointerMapping** subroutine, **XQueryKeymap** subroutine, **XSetPointerMapping** subroutine.

XModifierKeymap Data Structure

```
typedef struct {
   int max_keypermod;              /* Max number of keys per
                                      modifier of this server */
   KeyCode *modifiermap;           /* An 8 by max_keypermod array
                                      of the modifiers */
} XModifierKeymap;
```

Related Information

The **<X11/keysym.h>** header file, **<X11/keysymdef.h>** header file

The **XChangeKeyboardMapping** subroutine, **XDeleteModifiermapEntry** subroutine, **XFreeModifiermap** subroutine, **XGetKeyboardMapping** subroutine, **XGetModifierMapping** subroutine, **XInsertmodifiermapEntry** subroutine, **XLookupString** subroutine, **XNewModifiermap** subroutine, **XSetModifierMapping** subroutine.

XHostAddress Data Structure

```
typedef struct {
   int family;          /* for example FamilyInternet */
   int length;          /* length of address, in bytes */
   char *address;       /* pointer to where to find the address */
} XHostAddress;
```

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

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

length Specifies the length of the address in bytes.

address Specifies a pointer to the address.

Related Information

The **<X11/X.h>** header file, **/etc/X?.hosts** file.

The **XAddHost** subroutine, **XAddHosts** subroutine, **XListHosts** subroutine, **XRemoveHost** subroutine, **XRemoveHosts** subroutine.

XAnyEvent Data Structure

For each event type, a corresponding structure is declared in the **<X11/Xlib.h>** header file.

```
typedef struct {
   int type;
   unsigned long serial;     /* Number of last request processed
                                by the server */
   Bool send_event;          /* True if this came from a SendEvent
                                request */
   Display *display;         /* Display the event was read from */
   Window window;
} XAnyEvent:
```

All the event structures have the following common fields.

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 the value of **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.

Related Information

The **<X11/Xlib.h>** header file.

XEvent Data Structure

```
typedef union _XEvent {
    int type                    /* Must not be changed */
    XAnyEvent xany;
    XKeyEvent xkey;
    XButtonEvent xbutton;
    XMotionEvent xmotion;
    XCrossingEvent xcrossing;
    XFocusChangeEvent xfocus;
    XExposeEvent xexpose;
    XGraphicsExposeEvent xgraphicsexpose;
    XNoExposeEvent xnoexpose;
    XVisibilityEvent xvisibility;
    XCreateWindowEvent xcreatewindow;
    XDestroyWindowEvent xdestroywindow;
    XUnmapEvent xunmap;
    XMapEvent xmap;
    XMapRequestEvent xmaprequest;
    XReparentEvent xreparent;
    XConfigureEvent xconfigure;
     XGravityEvent xgravity;
     XResizeRequestEvent xresizerequest;
     XConfigureRequestEvent xconfigurerequest;
     XCirculateEvent xcirculate;
     XCirculateRequestEvent xcirculaterequest;
     XPropertyEvent xproperty;
     XSelectionClearEvent xselectionclear;
     XSelectionRequestEvent xselectionrequest;
     XSelectionEvent xselection;
     XColormapEvent xcolormap;
     XClientMessageEvent xclient;
     XMappingEvent xmapping;
     XErrorEvent xerror;
    XKeymapEvent xkeymap;
     long pad[24];
} XEvent;
```

Related Information

The **<X11/Xlib.h>** header file.

XButtonPressedEvent or XButtonReleasedEvent Data Structure

```
typedef struct {
    int type;                   /* ButtonPress or ButtonRelease */
    unsigned long serial;       /* Number of the last request
                                   processed by the server */
     Bool send_event;           /* True if this came from a
                                   SendEvent request */
     Display *display;          /* The display the event was read
                                   from */
     Window window;             /* The event window it is reported
                                   relative to */
     Window root;               /* Root window that the event
                                   occurred on */
     Window subwindow;          /* The child window */
     Time time;                 /* Milliseconds */
    int x, y;                   /* Pointer x, y coordinates in the
                                   event window */
    int x_root, y_root;         /* Coordinates relative to the root
                                   window */
    unsigned int state;         /* Key or button mask */
    unsigned int button;        /* Detail */
     Bool same_screen;          /* Same screen flag */
} XButtonEvent;
typedef XButtonEvent XButtonPressedEvent;
typedef XButtonEvent XButtonReleasedEvent;
```

The fields for these structures are defined as follows:

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* field set to **GraphicsExpose**.

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

send_event — Set to the value of **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.

window — 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.

root — The window ID of the root window of the source.

x_root — This is set to the x pointer coordinate relative to the origin of the root window at the time of the event.

y_root
: This is set to the y pointer coordinate relative to the origin of the root window at the time of the event.

same_screen
: Indicates if the event window is on the same screen as the root window. This parameter can be:

  TRUE If the event and root windows are on the same screen.

  FALSE If the event and root windows are not on the same screen.

subwindow
: 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 the value of **None**.

x
: Can be one of the following:

- If the event window is on the same screen as the root window, the x coordinate is set to the coordinate relative to the event window's origin
- Otherwise, *X* is the value of **0**.

y
: Can be one of the following:

- If the event window is on the same screen as the root window, the y coordinate is set to the coordinate relative to the event window's origin
- Otherwise, *Y* is the value of **0**.

time
: The time that the event was generated. The time is expressed in milliseconds since the server reset.

state
: 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 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**

button
: This represents the pointer buttons that changed state for the **XButtonPressedEvent** and **XButtonReleasedEvent** data structures, and can be set to one or more of the following button names: **Button1**, **Button2**, **Button3**, **Button4**, **Button5**.

Related Information

The **ButtonPress** protocol event, **ButtonRelease** protocol event.

XKeyPressedEvent or XKeyReleasedEvent Data Structure

```
typedef struct {
    int type;                   /* KeyPress or KeyRelease */
    unsigned long serial;       /* Number of the last request
                                   processed by the server */
    Bool send_event;            /* True if this came from a SendEvent
                                   request */
    Display *display;           /* The display the event was read
                                   from */
    Window window;              /* The event window it is reported
                                   relative to */
    Window root;                /* Root window that the event
                                   occurred on */
    Window subwindow;           /* The child window */
    Time time;                  /* Milliseconds */
    int x, y;                   /* Pointer x, y coordinates in the
                                   event window */
    int x_root, y_root;         /* Coordinates relative to the root
                                   window */
    unsigned int state;         /* Key or button mask */
    unsigned int keycode;       /* Detail */
    Bool same_screen;           /* Same screen flag */
} XKeyEvent;
typedef XKeyEvent XKeyPressedEvent;
typedef XKeyEvent XKeyReleasedEvent;
```

The fields for these structures are defined as follows:

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** data structure with the *type* field set to **GraphicsExpose**.

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

send_event — Set to the value of **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.

window — 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.

root — The window ID of the root window of the source.

x_root — This is set to the x pointer coordinate relative to the origin of the root window at the time of the event.

y_root — This is set to the y pointer coordinate relative to the origin of the root window at the time of the event.

same_screen Indicates if the event window is on the same screen as the root window. This field can be either of the following values:

TRUE If the event and root windows are on the same screen.

FALSE If the event and root windows are not on the same screen.

subwindow 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* field has the value of **None**.

x Can be one of the following:

- The x coordinate relative to the origin of the event window if the root window is on the same screen as the event window.
- Otherwise, the *x* field has the value of **0**.

y Can be one of the following:

- The y coordinate relative to the origin of the event window if the root window is on the same screen as the event window.
- Otherwise, the *y* field has the value of **0**.

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

state 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 include OR 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**

keycode This is set to a number that represents a physical key on the keyboard for the **XKeyPressedEvent** and **XKeyReleasedEvent** data structures.

Related Information

The **KeyPress** event, **KeyRelease** event.

XPointerMovedEvent Data Structure

```
typedef struct {
   int type;                    /* MotionNotify */
   unsigned long serial;        /* Number of the last request
                                   processed by the server */
    Bool send_event;            /* True if this came from a
                                   SendEvent request */
    Display *display;           /* The display the event was read
                                   from */
    Window window;              /* The event window it is reported
                                   relative to */
    Window root;                /* Root window that the event
                                   occurred on */
    Window subwindow;           /* The child window */
    Time time;                  /* Milliseconds */
   int x, y;                    /* Pointer x, y coordinates in the
                                   event window */
   int x_root, y_root;          /* Coordinates relative to the root
                                   window */
   unsigned int state;          /* Key or button mask */
   unsigned int keycode;        /* Detail */
   char is_hint;                /* Detail */
    Bool same_screen;           /* Same screen flag */
} XMotionEvent;
typedef XMotionEvent XPointerMovedEvent;
```

The fields for the **XPointerMovedEvent** data structure are defined as follows:

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* field set to the value of **GraphicsExpose**.

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

send_event — Set to the value of **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.

window — 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.

root — The window ID of the root window of the source.

x_root — This is set to the x pointer coordinate relative to the origin of the root window at the time of the event.

y_root — This is set to the y pointer coordinate relative to the origin of the root window at the time of the event.

same_screen Indicates if the event window is on the same screen as the root window. This field can be either of the following values:

TRUE If the event and root windows are on the same screen.

FALSE If the event and root windows are not on the same screen.

subwindow 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* field has the value of **None**.

x Can be one of the following:

- The x coordinate relative to the origin of the event window if the source window and the event window are on the same screen.
- Otherwise, the *x* field has the value of **0**.

y Can be one of the following:

- The y coordinate relative to the origin of the event window if the source window and the event window are on the same screen.
- Otherwise, the *y* field has the value of **0**.

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

state 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 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**

is_hint This can be set to the value of **NotifyNormal** or **NotifyHint** for the **XPointerMovedEvent** data structure.

Related Information

The **MotionNotify** event.

XCrossingEvent or XEnterWindowEvent or XLeaveWindowEvent Data Structures

```
typedef struct {
    int type;                       /* EnterNotify or LeaveNotify */
    unsigned long serial;           /* Number of the last request
                                       processed by the server */
     Bool send_event;               /* True if this came from a
                                       SendEvent request */
     Display *display;              /* The display the event was read
                                       from */
     Window window;                 /* The event window it is
                                       reported relative to */
    Window root;                    /* Root window that the event
                                       occurred on */
     Window subwindow;              /* The child window */
     Time time;                     /* Milliseconds */
    int x, y;                       /* Pointer x, y coordinates in
                                       the event window */
    int x_root, y_root;             /* Coordinates relative to the
                                       root window */
    int mode;                       /* NotifyNormal, NotifyGrab,
                                       NotifyUngrab */
    int detail;                     /* NotifyAncestor, NotifyVirtual,
                                       NotifyInferior,
                                       NotifyNonlinear,
                                       NotifyNonlinearVirtual */
     Bool same_screen;              /* Same screen flag */
     Bool focus;                    /* Boolean focus */
    unsigned int state;             /* Key or button mask */
} XCrossingEvent;
typedef XCrossingEvent XEnterWindowEvent;
typedef XCrossingEvent XLeaveWindowEvent;
```

The fields of the **XCrossingEvent**, **XEnterWindowEvent**, and **XLeaveWindowEvent** data structures are as follows:

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* field set to **GraphicsExpose**.
display	Set to a pointer to the display the event was read on.
send_event	Set to the value of **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.
window	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.
root	The ID of the root window on which the event occurred.

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

subwindow 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 the value of **None**.

time The time (in milliseconds) the event was generated.

x The x pointer position in the event window. This position is always the final position of the pointer, not the initial position of the pointer.

y The y 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, the *x* and *y* fields are set to the value of **0**.

x_root Set to the x pointer coordinate relative to the origin of the root window at the time of the event.

y_root Set to the y pointer coordinate relative to the origin of the root window at the time of the event.

same_screen Indicates if the event window is on the same screen as the root window. The *same_screen* field can be either of the following values:

True The event and root windows are on the same screen.

False The event and root windows are not on the same screen.

focus Indicates whether the event window is the focus window or an inferior of the focus window. The *focus* field can be either of the following values:

True The event window is the focus window or an inferior of the focus window.

False The event window is neither the focus window nor an inferior of the focus window.

state Indicates the state of the pointer buttons and modifier keys immediately prior the event. The X Server can set this field to the bitwise-inclusive OR of one or more of the following button or modifier key mask values.

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

mode Indicates if the events are normal events or pseudo motion events when a grab activates, or when a grab deactivates. The X Server can set the *mode* field to one of the following values:

NotifyNormal

NotifyGrab

NotifyUngrab.

detail Indicates the notify detail can be set to one of the following values:

NotifyAncestor **NotifyVirtual**

NotifyInferior **NotifyNonlinear**

NotifyNonlinearVirtual

Related Information

The **EnterNotify** event, **LeaveNotify** event.

The **XChangeActivePointerGrab** subroutine, **XGrabKeyboard** subroutine, **XGrabPointer** subroutine, **XUngrabPointer** subroutine.

XFocusChangeEvent or XFocusInEvent or XFocusOutEvent Data Structures

```
typedef struct {
    int type;                       /* FocusIn or FocusOut */
    unsigned long serial;           /* Number of the last request
                                       processed by the server */
    Bool send_event;                /* True if this came from a
                                       SendEvent request */
     Display *display;              /* The display the event was read
                                       from */
     Window window;                 /* The window of the event */
    int mode;                       /* NotifyNormal, NotifyGrab,
                                       NotifyUngrab */
    int detail;                     /* NotifyAncestor, NotifyVirtual,
                                       NotifyInferior,
                                       NotifyNonlinear,
                                       NotifyNonlinearVirtual,
                                       NotifyPointer,
                                       NotifyPointerRoot,
                                       NotifyDetailNone */
} XFocusChangeEvent;
typedef XFocusChangeEvent XFocusInEvent;
typedef XFocusChangeEvent XFocusOutEvent;
```

The fields of the **XFocusChange**, **XFocusInEvent** and **XFocusOutEvent** data structures include the following definitions:

window — 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.

mode — Specifies the type of focus event. The *mode* field can be set to one of the following values:

NotifyNormal	Specifies a normal focus event.
NotifyWhileGrabbed	Specifies a focus event while grabbed.
NotifyGrab	Specifies a focus event when a grab activates.
NotifyUngrab	Specifies a focus event when a grab deactivates.

detail Indicates the notify detail depending on the event mode. The *detail* field can be one of the following values:

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.

Related Information

The **FocusIn** event, **FocusOut** event.

The **XGrabKeyboard** subroutine, **XUngrabKeyboard** subroutine.

XKeymapEvent Data Structure

```
typedef struct {
    int type;                       /* KeymapNotify */
    unsigned long serial;           /* Number of the last request
                                       processed by the server */
    Bool send_event;                /* True if this came from a
                                       SendEvent request */
    Display *display;               /* The display the event was
                                       read from */
    Window window;
    char key_vector[32];
} XKeymapEvent;
```

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

window — This is not used, but present for use with toolkit operations.

key_vector — 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`.

Related Information

The **KeymapNotify** event.

XExposeEvent Data Structure

```
typedef struct {
    int type;                          /* Expose */
    unsigned long serial;              /* Number of the last request
                                          processed by the server */
    Bool send_event;                   /* True if this came from a
                                          SendEvent request */
    Display *display;                  /* The display the event was read
                                          from */
    Window window;
    int x, y;
    int width, height:
    int count;                         /* If nonzero, at least this many
                                          more */
} XExposeEvent;
```

The fields of the **XExposeEvent** data structure include the following:

window — The window ID of the exposed (damaged) window.

x — Indicates the x coordinate of the rectangle. This coordinate is set relative to the origin of the drawable.

y — Indicates the y coordinate of the rectangle. This coordinate is set relative to the origin of the drawable.

width — Specifies the width extent of the rectangle.

height — Specifies the height extent of the rectangle.

count — Specifies the number of **Expose** events that should follow. The *count* field can be:

- 0 — No **Expose** events will follow. (Applications not designed to optimize re display by distinguishing between subareas of a window re display entirely if the *count* field is the value of **0**.)
- 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 the *count* field is a nonzero value.)

Related Information

The **Expose** event.

XGraphicsExposeEvent Data Structure

```
typedef struct {
   int type;                    /* GraphicsExpose */
   unsigned long serial;        /* Number of the last request
                                   processed by the server */
    Bool send_event;            /* True if this came from a
                                    SendEvent request */
    Display *display;           /* The display the event was read
                                   from */
    Drawable drawable;
   int x, y;
    int width, height:
    int count;                  /* If nonzero, at least this many
                                   more */
    int major_code;             /* core is CopyArea or CopyPlane */
    int minor_code;             /* Not defined in the core */
} XGraphicsExposeEvent;
```

The **XGraphicsExposeEvent** data structure included the following fields:

drawable Specifies the drawable ID of the destination region on which the graphics request is to be performed.

major_code Specifies the graphics request initiated by the client This field can have one of the following values:

X_CopyArea	Indicates that a call to the **XCopyArea** subroutine initiated the request.
X_CopyPlane	Indicates that a call to the **XCopyPlane** subroutine initiated the request.

These constants are defined in the **<X11/Xproto.h>** file.

minor_code Specifies the graphics request initiated by the client. This field, however, is not defined by the core X protocol and will have the value of **0** in these cases, although it may be used as an extension.

x Specifies the x coordinate of the upper left corner of the rectangle. This coordinate is relative to the origin of the drawable.

y Specifies the y coordinate of the upper left corner of the rectangle. This coordinate is relative to the origin of the drawable.

width Specifies the size (extent) of the rectangle.

height Specifies the size (extent) of the rectangle.

count Specifies the number of **GraphicsExpose** events to follow for the specified window. This field can have the following values:

0	Indicates that no **GraphicsExpose** events will follow.
nonzero	Indicates that at least that number, and possibly more, **GraphicsExpose** events will follow.

Related Information

The **GraphicsExposure** event, **NoExpose** event.

The **XCopyArea** subroutine, **XCopyPlane** subroutine, **XCreateGC** subroutine, **XSetGraphicsExposures** subroutine.

XNoExposeEvent Data Structure

```
typedef struct {
    int type;                   /* NoExpose */
    unsigned long serial;       /* Number of the last request
                                   processed by the server */
     Bool send_event;           /* True if this came from a SendEvent
                                   request */
     Display *display;          /* The display the event was read
                                   from */
    Drawable drawable;
    int major_code;             /* core is CopyArea or CopyPlane */
    int minor_code;             /* Not defined in the core */
} XNoExposeEvent;
```

The **XGraphicsExposeEvent** and **XNoExposeEvent** data structures have the following common fields:

drawable Specifies the drawable ID of the destination region on which the graphics request is to be performed.

major_code Specifies the graphics request initiated by the client The *major_code* field can have either of the following values:

X_CopyArea Indicates that a call to the **XCopyArea** subroutine initiated the request.

X_CopyPlane Indicates that a call to the **XCopyPlane** subroutine initiated the request.

minor_code Specifies the graphics request initiated by the client. The *minor_code* field, however, is not defined by the core X protocol and will have the value of **0** in these cases, although it may be used as an extension.

Related Information

The **<X11/Xproto.h>** file.

The **GraphicsExpose** event, **NoExpose** event.

The **XCopyArea** subroutine, **XCopyPlane** subroutine, **XCreateGC** subroutine, **XSetGraphicsExposures** subroutine.

XCirculateEvent Data Structure

```
typedef struct {
   int type;                /* CirculateNotify */
   unsigned long serial;    /* Number of last request processed
                               by the server */
   Bool send_event;         /* True if this came from a SendEvent
                               request */
    Display *display;        /* The display the event was read
                               from */
    Window event;
    Window window;
   int place;               /* PlaceOnTop, PlaceOnBottom */
} XCirculateEvent;
```

The **XCirculateEvent** data structure includes the following fields:

type Specifies the event type, **CirculateNotify.**

serial Specifies the number of the last request processed by the server.

send_event Specifies **True** if this came from a SendEvent request.

display Specifies the display that the event was read from.

event Specifies the window on which the event was generated. This field is set to either the restacked window or its parent, depending on whether **StructureNotify** or **SubstructureNotify** was selected.

window Specifies the window that was restacked.

place Specifies the window position after the restack occurs. The *place* field can have either of the following values:

PlaceOnTop Indicates that the window is now on top of all siblings.

PlaceOnBottom Indicates that the window is now below all siblings.

Related Information

The **CirculateNotify** event.

The **XCirculateSubwindows** subroutine, **XCirculateSubwindowsDown** subroutine, **XCirculateSubwindowsUp** subroutine.

XConfigureEvent Data Structure

```
typedef struct {
   int type;                    /* ConfigureNotify */
   unsigned long serial;        /* Number of the last request
                                   processed by the server */
    Bool send_event;             /* True if this came from a
                                   SendEvent request */
    Display *display;            /* The display the event was read
                                   from */
    Window event:
    Window window;
   int x, y;
   int width, height;
    int border_width;
    Window above;
    Bool override_redirect;
} XConfigureEvent;
```

The **XConfigureEvent** data structure includes the following fields:

event — Specifies the window on which the event was genereated. This field is set to either the reconfigured window or its parent, depending on whether **StructureNotify** or **SubstructureNotify** was selected.

window — Specifies the window whose size, position, border, or stacking order was changed.

x — Specifies the x coordinate of the upper left corner of the window. This coordinate is relative to the origin of the new parent window.

y — Specifies the y coordinate of the upper left corner of the window. This coordinate is relative to the origin of the new parent window

width — Specifies the size of the window, excluding the border.

height — Specifies the size of the window, excluding the border.

border_width — Specifies the width of the window border, in pixels.

above — Specifies the window ID of the sibling window. This field is used for stacking operations.

If the *above* field is set to the value of **None**, the reconfigured window is on the bottom of the stack with respect to sibling windows.

If this field is set to a sibling window, the reconfigured window is placed on top of this sibling window.

override_redirect — Specifies the value set in the *override_redirect* field of the window. If this field is the value of **True**, window manager clients should ignore the window.

Related Information

The **ConfigureNotify** event.

The **XConfigureWindow** subroutine, **XLowerWindow** subroutine, **XRaiseWindow** subroutine, **XRestackWindows** subroutine, **XMoveWindow** subroutine, **XResizeWindow** subroutine, **XMoveResizeWindow** subroutine, **XMapRaised** subroutine, **XSetWindowBorderWidth** subroutine.

XCreateWindowEvent Data Structure

The **XCreateWindowEvent** data structure includes the following fields:

parent	Specifies the parent of the created window.
window	Specifies the created window.
x	Specifies the x coordinate of the upper left outside corner of the created window. This coordinate is relative to the inside of the borders of the parent window.
y	Specifies the y coordinate of the upper left outside corner of the created window. This coordinate is relative to the inside of the borders of the parent window.
width	Specifies the inside size of the created window, excluding the border. This field is always a nonzero value.
height	Specifies the inside size of the created window, excluding the border. This field is always a nonzero value.
border_width	Specifies the width of the border of the created window, in pixels.
override_redirect	Specifies the value set in the *override_redirect* field of the window. If this field is set to the value of **True**, window manager clients should ignore the window.

Related Information

The **CreateNotify** event.

The **XCreateSimpleWindow** subroutine, **XCreateWindow** subroutine.

XDestroyWindowEvent Data Structure

```
typedef struct {
   int type;                  /* DestroyNotify */
   unsigned long serial;      /* Number of the last request
                                 processed by the server */
    Bool send_event;          /* True if this came from a SendEvent
                                 request */
    Display *display;         /* The display the event was read
                                 from */
    Window event;
    Window window;
} XDestroyWindowEvent;
```

The **XDestroyWindowEvent** data structure includes the following fields:

event Specifies the window on which the event was generated. This field is set to either the destroyed window or its parent, depending on whether **StructureNotify** or **SubstructureNotify** was selected.

window Specifies the window that is destroyed.

Related Information

The **DestroyNotify** event.

The **XDestroySubwindows** subroutine, **XDestroyWindow** subroutine.

XGravityEvent Data Structure

```
typedef struct {
   int type;                  /* GravityNotify */
   unsigned long serial;      /* Number of the last request
                                 processed by the server */
    Bool send_event;          /* True if this came from a
                                 SendEvent request */
    Display *display;         /* The display the event was read
                                 from */
    Window event;
    Window window;
   int x, y;
} XGravityEvent;
```

The **XGravityEvent** data structure includes the following fields:

event Specifies the window on which the event was generated. This field can be set to either the window that was moved or its parent, depending on whether **StructureNotify** or **SubstructureNotify** was selected.

window Specifies the window that was moved.

x Specifies the x coordinate of the upper left outside corner of the window. This coordinate is relative to the origin of the new parent window.

y Specifies the y coordinate of the upper left outside corner of the window. This coordinate is relative to the origin of the new parent window.

Related Information

The **GravityNotify** event.

The **XConfigureWindow** subroutine, **XMoveResizeWindow** subroutine, **XResizeWindow** subroutine.

XMapEvent Data Structure

```
typedef struct {
   int type;                    /* MapNotify*/
   unsigned long serial;        /* Number of the last request
                                   processed by the server */
    Bool send_event;            /* True if this came from a
                                   SendEvent request */
    Display *display;           /* The display the event was read
                                   from */
    Window event;
    Window window;
    Bool override_redirect;     /* Boolean, is override set... */
} XMapEvent;
```

The **XMapEvent** data structure includes the following fields:

event Specifies the window on which the event was generated. This field is set to either the mapped window or its parent, depending on whether **StructureNotify** or **SubstructureNotify** was selected.

window Specifies the window that was mapped.

override_redirect Specifies the value set in the *override_redirect* field of the window. If this field is the value of **True**, window manager clients should ignore this window, because these events usually are generated from pop ups, which override structure control.

Related Information

The **MapNotify** event.

The **XMapRaised** subroutine, **XMapSubwindows** subroutine, **XMapWindow** subroutine.

XMappingEvent Data Structure

```
typedef struct {
   int type;                  /* MappingNotify */
   unsigned long serial;      /* Number of the last request
                                 processed by the server */
    Bool send_event;          /* True if this came from a SendEvent
                                 request */
    Display *display;         /* The display the event was read
                                 from */
    Window window;            /* Unused */
    int request;              /* MappingModifier, MappingKeyboard,
                                       MappingPointer */
    int first_keycode;        /* The first_keycode */
    int count;                /* Defines the range of change with
                                 first_keycode*/
} XMappingEvent;
```

The **XMappingEvent** data structure includes the following fields:

request Specifies the kind of mapping change that occurred. The *request* field can have the following values:

MappingModifier	Indicates that the modifier mapping was changed.
MappingKeyboard	Indicates that the keyboard mapping was changed.
MappingPointer	Indicates that the pointer button mapping was changed.

first_keycode Specifies the first number in the range of the altered mapping. This field is set only if the *request* field is **MappingKeyboard**.

count Specifies the number of keycodes altered. This field is set only if the *request* field is **MappingKeyboard**.

To update the client application's knowledge of the keyboard, use the **XRefreshKeyboardMapping** subroutine.

Related Information

The **MappingNotify** event.

The **XChangeKeyboardMapping** subroutine, **XRefreshKeyboardMapping** subroutine, **XSetModifierMapping** subroutine, **XSetPointerMapping** subroutine.

XReparentEvent Data Structure

```
typedef struct {
    int type;                   /* ReparentNotify */
    unsigned long serial;       /* Number of the last request
                                   processed by the server */
    Bool send_event;            /* True if this came from a
                                   SendEvent request*/
    Display *display;           /* The display the event was read
                                   from */
    Window event;
    Window window;
    Window parent;
    int x, y;
    Bool override_redirect;
} XReparentEvent;
```

The **XReparentEvent** data structure includes the following fields:

event	Specifies the window on which the event was generated. This field is set to either the reparented window or the old or new parent, depending on whether **StructureNotify** or **SubstructureNotify** was selected.
window	Specifies the window that was reparented.
parent	Specifies the new parent window.
x	Specifies the x coordinate of the upper left outer corner of the reparented window. This coordinate is relative to the origin of the new parent window.
y	Specifies the y coordinate of the upper left outer corner of the reparented window. This coordinate is relative to the origin of the new parent window
override_redirect	Specifies the value set in the *override_redirect* field of the reparented window. If this field is the value of **True**, window manager clients should ignore the window.

Related Information

The **ReparentNotify** event.

The **XReparentWindow** subroutine.

XUnmapEvent Data Structure

```
typedef struct {
   int type;                  /* UnmapNotify */
   unsigned long serial;      /* Number of the last request
                                 processed by the server */
    Bool sendevent;           /* True if this came from a
                                 SendEvent request */
    Display *display;         /* The display the event was read
                                 from */
    Window event;
    Window window;
    Bool from_configure;
} XUnmapEvent;
```

The **XUnmapEvent** data structure includes the following fields:

event	Specifies the window on which this event was generated. This paramer may be set to either the unmapped window or its parent, depending on whether **StructureNotify** or **SubstructureNotify** was selected.
window	Specifies the window that was unmapped.
from_configure	Specifies the value of **True** if the event was generated as a result of resizing the parent window when the window itself had a *win_gravity* field of **UnmapGravity**.

Related Information

The **UnmapNotify** event.

The **XUnmapSubwindows** subroutine, **XUnmapWindow** subroutine.

XVisibilityEvent Data Structure

```
typedef struct {
   int type;                    /* VisibilityNotify */
   unsigned long serial;        /* Number of the last request
                                   processed by the server */
   Bool send_event;             /* True if this came from a
                                   SendEvent request */
   Display *display;            /* The display the event was read
                                   from */
   Window window;
   int state;
} XVisibilityEvent;
```

The **XVisibilityEvent** data structure includes the following fields:

window Specifies the window whose visibility state changes.

state Specifies the visibility state of the window. This field can have one of the following values:

VisibilityUnobscured

VisibilityPartiallyObscured

VisibilityFullyObscured.

Related Information

The **VisibilityNotify** event.

XCirculateRequestEvent Data Structure

```
typedef struct {
   int type;                /* CirculateRequest */
   unsigned long serial;    /* Number of the last request
                               processed by the server */
   Bool send_event;         /* True if this came from a
                               SendEvent request */
   Display *display;        /* The display the event was read
                               from */
   Window parent;
   Window window;
   int place;               /* PlaceOnTop, PlaceOnBottom */
} XCirculateRequestEvent;
```

The **XCirculateRequestEvent** data structure includes the following fields:

parent Specifies the parent window.

window Specifies the subwindow to be restacked.

place Specifies the new position of the window in the stacking order. This field can have either of the following values:

PlaceOnTop Indicates that the window will be placed on top of all siblings.

PlaceOnBottom Indicates that the window will be placed below all siblings.

Related Information

The **CirculateRequest** event.

The **XCirculateSubwindows** subroutine, **XCirculateSubwindowsDown** subroutine, **XCirculateSubwindowsUp** subroutine.

XConfigureRequestEvent Data Structure

```
typedef struct {
   int type;                  /* ConfigureRequest */
   unsigned long serial;      /* Number of the last request
                                 processed by the server */
    Bool send_event;          /* True if this came from a
                                 SendEvent request */
    Display *display;         /* The display the event was read
                                 from */
    Window parent;
    Window window;
   int x, y;
   int width, height;
   int border_width;
    Window above;
   int detail;                /* Above, Below, TopIf, BottomIf,
                                    Opposite */
   unsigned long value_mask;
} XConfigureRequestEvent;
```

The **XConfigureRequestEvent** data structure includes the following fields:

parent Specifies the parent window.

window Specifies the window to be reconfigured.

x Specifies the x coordinate of the upper left outer corner of the reconfigured window. The value for this field is set according to the current geometry of the window.

y Specifies the y coordinate of the upper left outer corner of the reconfigured window. The value for this field is set according to the current geometry of the window.

width Specifies the size of the reconfigured window, excluding the border. The value for this field is set according to the current geometry of the window.

height Specifies the size of the reconfigured window, excluding the border. The value for this field is set according to the current geometry of the window.

border_width Specifies the width of the border of the reconfigured window, in pixels. The value for this field is set according to the current geometry of the window.

above Specifies the sibling window. This field can have one of the following values:

None Indicates that the reconfigured window is placed on the bottom of the stack with respect to sibling windows. This is the default value for this field.

SiblingWindow IDs The reconfigured window is placed on top of these sibling windows.

detail	Specifies the notify detail. This member can be set to **Below**, **TopIf**, **BottomIf**, or **Opposite**. The default value for this field is **Above**.
value_mask	Specifies which components were indicated in the configure window request.

Related Information

The **ConfigureRequest** event.

The **XConfigureWindow** subroutine, **XLowerWindow** subroutine, **XMapRaised** subroutine, **XMoveResizeWindow** subroutine, **XMoveWindow** subroutine, **XRaiseWindow** subroutine, **XResizeWindow** subroutine, **XRestackWindows** subroutine, **XSetWindowBorderWidth** subroutine.

XMapRequestEvent Data Structure

```
typedef struct {
   int type;                /* MapRequest */
   unsigned long serial;    /* Number of the last request
                               processed by the server */
    Bool send_event;        /* True if this came from a
                               SendEvent request */
    Display *display;       /* The display the event was read
                               from */
   Window parent;
    Window window;
} XMapRequestEvent;
```

The **XMapRequestEvent** data structure includes the following fields:

parent	Specifies the parent window.
window	Specifies the window to be mapped.

Related Information

The **MapRequest** event.

The **XMapRaised** subroutine, **XMapSubwindows** subroutine, **XMapWindow** subroutine.

XResizeRequestEvent Data Structure

```
typedef struct {
    int type;                   /* ResizeRequest */
    unsigned long serial;       /* Number of the last request
                                   processed by the server */
    Bool send_event;            /* True if this came from a
                                   SendEvent request */
    Display *display;           /* The display the event was read
                                   from */
    Window parent;
    int width, height;
} XResizeRequestEvent;
```

The **XResizeRequestEvent** data structure includes the following fields:

window	Specifies the window that another client attempted to resize.
width	Specifies the inside size of the window, excluding the border.
height	Specifies the inside size of the window, excluding the border.

Related Information

The **ResizeRequest** event.

The **XConfigureWindow** subroutine, **XMoveResizeWindow** subroutine, **XResizeWindow** subroutine.

XColormapEvent Data Structure

```
typedef struct {
    int type;                       /* ColormapNotify */
    unsigned long serial;           /* Number of the last request
                                       processed by the server */
    Bool send_event;                /* True if this came from a
                                       SendEvent request */
    Display *display;               /* The display the event was read
                                       from */
    Window window;
    Colormap colormap;              /* The colormap or None */
    Bool new;
    int state;                     /* ColormapInstalled,
                                       ColormapUninstalled*/
} XColormapEvent;
```

The **XColormapEvent** data structure includes the following fields:

window Specifies the window whose associated colormap is changed, installed, or uninstalled.

colormap Specifies the colormap associated with the window for a colormap changed by a call to the **XChangeWindowAttributes** subroutine. For a colormap changed by a call to the **XFreeColormap** subroutine, this field is the value of **None**.

new Specifies if the colormap for the specified window was changed or installed or uninstalled. This field can have either of the following values:

True Indicates that the colormap was changed.

False Indicates that the colormap was installed or uninstalled.

state Specifies if the colormap is installed or uninstalled. This field can have either of the following values:

ColormapInstalled

ColormapUninstalled

Related Information

The **ColormapNotify** event.

The **XChangeWindowAttributes** subroutine, **XFreeColormap** subroutine, **XInstallColormap** subroutine, **XSetWindowColormap** subroutine, **XUninstallColormap** subroutine.

XClientMessageEvent Data Structure

```
typedef struct {
    int type;                       /* ClientMessage */
    unsigned long serial;           /* Number of the last request
                                       processed by the server */
    Bool send_event;                /* True if this came from a
                                       SendEvent request */
    Display *display;               /* The display the event was read
                                       from */
    Window window;
    Atom message_type;
    int format;
    union {
        char b[20];
        short s[10];
        long l[5];
        } data;
} XClientMessageEvent;
```

The **XClientMessageEvent** data structure includes the following fields:

window	Specifies the window to which the event was sent.
message_type	Specifies an atom which indicates how the data is to be interpreted by the receiving client. This field is not interpreted by the X Server.
format	Specifies whetner the data should be viewed as a list of bytes, shorts, or longs. This field should be set to 8 bits, 16 bits, or 32 bits.
data	Specifies a union which contains the members b (bytes), s (shorts), and l (longs). These members represent data of 20 8-bit values,10 16-bit values, and 5 32-bit values. Some message types may not use all these values. This field is not interpreted by the X Server.

Related Information

The **XAnyEvent** data structure.

The **ClientMessage** event.

The **XSendEvent** subroutine.

XPropertyEvent Data Structure

```
typedef struct {
    int type;                       /* PropertyNotify */
    unsigned long serial;           /* Number of the last request
                                       processed by the server */
    Bool send_event;                /* True if this came from a
                                       SendEvent request */
    Display *display;               /* The display the event was
                                       read from */
    Window window;
    Atom atom;
    Time time;
    int state;                      /* PropertyNewValue,
                                              PropertyDeleted*/
} XPropertyEvent;
```

The **XPropertyEvent** data structure includes the following fields:

window Specifies the window whose associated property is changed.

atom Specifies the atom of the property that is changed or requested.

time Specifies the server time when the property is changed.

state Specifies whether the property is changed to a new value or deleted. This field can have the following values:

PropertyNewValue Indicates that the property is changed or that it is replaced with identical data using the **XChangeProperty** or **XRotateWindowProperties** subroutines.

PropertyDeleted Indicates that the property is deleted using the **XDeleteProperty** or **XGetWindowProperty** subroutines.

Related Information

The **PropertyNotify** event.

The **XChangeProperty** subroutine, **XDeleteProperty** subroutine, **XGetWindowProperty** subroutine, **XRotateWindowProperties** subroutine.

XSelectionClearEvent Data Structure

```
typedef struct {
   int type;                    /* SelectionClear */
   unsigned long serial;        /* Number of the last request
                                   processed by the server */
    Bool send_event;            /* True if this came from a
                                   SendEvent request */
   Display *display;            /* The display the event was read
                                   from */
   Window window;
   Atom selection;
   Time time;
} XSelectionClearEvent;
```

The **XSelectionClearEvent** data structure includes the following fields:

window	Specifies the window losing ownership of the selection.
selection	Specifies the selection atom.
time	Specifies the last change time recorded for the selection.

Related Information

The **SelectionClear** event.

The **XSetSelectionOwner** subroutine.

XSelectionRequestEvent Data Structure

```
typedef struct {
   int type;                    /* SelectionRequest */
   unsigned long serial;        /* Number of the last request
                                   processed by the server */
    Bool send_event;            /* True if this came from a
                                   SendEvent request */
    Display *display;           /* The display the event was read
                                   from */
    Window owner;
    Window requestor;
    Atom selection;
    Atom target;
    Atom property;
    Time time;
} XSelectionRequestEvent;
```

The **XSelectionRequestEvent** data structure includes the following fields:

owner	Specifies the window owning the selection. This is the window specified by the current owner in the **XSetSelectionOwner** subroutine.
requestor	Specifies the window requesting the selection.
selection	Specifies the atom that names the selection.
target	Specifies the atom which indicates the type the selection is requested in.
property	Specifies a property name or the value of **None**.
time	Specifies the time, either in a timestamp (milliseconds) or in **CurrentTime**, taken from the **XConvertSelection** request.

Related Information

The **SelectionRequest** event.

The **XConvertSelection** subroutine, **XSetSelectionOwner** subroutine.

XSelectionEvent Data Structure

```
typedef struct {
    int type;                          /* SelectionNotify */
    unsigned long serial;              /* Number of the last request
                                          processed by the server */
    Bool send_event;                   /* True if this came from a
                                          SendEvent request */
    Display *display;                  /* The display the event was
                                          read from */
    Window requestor;
    Atom selection;
    Atom target;
    Atom property;                     /* The atom or None */
    Time time;
} XSelectionEvent;
```

The **XSelectionEvent** data structure includes the following fields:

requestor	Specifies the window associated with the requestor of the selection.
selection	Specifies the atom that indicates the selection.
target	Specifies the atom that indicates the converted type.
property	Specifies the atom that indicates the property the result is stored on. This field is set to the value of **None** if the conversion fails.
time	Specifies the time when the conversion took place. This can be a timestamp (in milliseconds) or **CurrentTime**.

Related Information

The **SelectionNotify** event.

The **XConvertSelection** subroutine, **XSendEvent** subroutine.

XErrorEvent Data Structure

```
typedef struct {
   int type;
    Display *display;              /* The display the event was
                                      read from */
   unsigned long serial;          /* The serial number of the
                                      failed request */
   unsigned char error_code;      /* The error code of the failed
                                      request */
   unsigned char request_code;    /* The major op code of the
                                      failed request */
   unsigned char minor_code;      /* The minor op code of the
                                      failed request */
    XID resourceid;               /* The resource id */
} XErrorEvent;
```

The **XErrorEvent** data structure includes the following fields:

display	Specifies the display that the event was read from.
serial	Specifies the number of requests, starting with the one sent over the network connection when it was opened. This value is the value of **NextRequest** immediately before the failing call was made.
error_code	Specifies the error code of the failed request.
request_code	Specifies a protocol request of the procedure that failed. The *request_code* field values are defined in the **<X11/Xproto.h>** file.
minor_code	Specifies the minor op code of the failed request.
resourceid	Specifies the resource ID.

Related Information

The **<X11/Xproto.h>** header file.

The **XDisplayName** subroutine, **XGetErrorDatabaseText** subroutine, **XGetErrorText** subroutine, **XSetErrorHandler** subroutine, **XSetIOErrorHandler** subroutine.

XWMHints Data Structure

```
#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   AllHints           (InputHint/StateHint/IconPixmapHint/
                              IconWindowHint/IconPositionHint/
                              IconMaskHint/WindowGroupHint)


typedef struct {
   long flags;              /* Marks which fields in this
                               structure are defined */
    Bool input;              /* Indicates whether this application
                               relies on the window manager to get
                               keyboard input */
   int initial_state;       /* The initial state of the
                               application */
    Pixmap icon_pixmap;      /* The pixmap to be used as the icon */
    Window icon_window;      /* The window to be used as the icon */
    int icon_x, icon_y;      /* The initial position of the icon */
    Pixmap icon_mask;        /* The pixmap to be used as the mask
                               for the icon_pixmap field */
    XID window_group;        /* The id of the related window
                               group */
} XWMHints ;
```

The **XWMHints** data structure includes the following fields:

flags Specifies which fields are defined in the **XWMHints** data structure. The values for this field are as follows:

InputHint

StateHint

IconPixmapHint

IconWindowHint

IconPositionHint

IconMaskHint

WindowGroupHint

AllHints

input Specifies the input focus model used by the application. This field communicates the input focus model to the window manager and can have the following values:

True Indicates that the application accepts input, but never explicitly sets focus to any of the subwindows. These applications use the push model of focus management. The *input* field also has this value if the application sets input focus to its subwindows only when it is given to its top level window by a window manager.

False Indicates that the application manages its input focus by explicitly setting focus to one of its subwindows whenever keyboard input is requested. These applications use the pull model of focus management. The *input* field also has this value if the application never expects any keyboard input.

Pull model window managers should make it possible for push model application to get input by setting input focus to the top level windows of applications with the *input* field set to the value of **True**. Push model window managers should ensure that pull model applications do not break them by resetting the input focus to **PointerRoot** when it is appropriate.

initial_state Specifies the initial state of the application. The values for this field are:

DontCareState Don't know or care

NormalState Most applications start this way

ZoomState The application wants to start zoomed

IconicState The application wants to start as an icon

InactiveState The application believes it is seldom used; some window managers may put it on inactive menu

icon_mask Specifies which pixels of the *icon_pixmap* field should be used as the icon. The *icon_mask* field allows for nonrectangular pixmaps. Both fields must be bit maps.

icon_window Specifies the window to be used as an icon for window managers that support such use.

window_group Specifies if this window belongs to a group of other windows. For example, if a single application manipulates multiple top level windows, this field provides the window manager with enough information to iconify all of the windows instead of only one window.

Related Information

The **XGetWMHints** subroutine, **XSetWMHints** subroutine.

XSizeHints Data Structure

```
#define   USPosition    (1L<<0)   /* user specified x, y */
#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   PResizeInc    (1L<<6)   /* program specified resize
                                     increments */
#define   PAspect       (1L<<7)   /* program specified min and
                                     max aspect ratios */
#define   PAllHints               (PPosition/PSize/PMinSize/
                                  PMaxSize/PResizeInc/PAspect)


typedef struct {
   long flags;                    /* Marks which fields in this
                                     structure are defined */
   int x, y;
   int width, height;
   int min_width, min_height;
   int max_width, max_height;
   int width_inc, height_inc;
   struct {
      int x;                      /* The numerator */
      int y;                      /* The denominator */
   } min_aspect, max_aspect;
   int base_width, base_height;
   int win_gravity;
} XSizeHints
```

The **XSizeHints** data structure includes the following fields:

flags Specifies how the position and size of the window is set. The values for this field are as follows:

USPosition	User specified x, y
USSize	User specified width, height
PPosition	Program specified position
PSize	Program specified size
PMinSize	Program specified minimum size
PMaxSize	Program specified maximum size
PResizeInc	Program specified resize increments
PAspect	Program specified minimum and maximum aspect ratios

	PAllHints (PPosition|PSize|PMinSize|PMaxSize|PResizeInc|PAspect)
x	Specifies the x coordinate for the upper left corner of the window.
y	Specifies the y coordinate for the upper left corner of the window.
width	Specifies the width of the window.
height	Specifies the height of the window.
min_width	Specifies the minimum width of the window for the application.
min_height	Specifies the minimum height of the window for the application.
max_width	Specifies the maximum width of the window.
max_height	Specifies the maximum height of the window.
width_inc	Specifies an arithmetic progression of sizes, from minimum size to maximum size, for the window resize requests.
height_inc	Specifies an arithmetic progression of sizes, from minimum size to maximum size, for the window resize requests.
min_aspect	Specifies the minimum of the range of aspect ratios the application prefers. This field is expressed as a ratio of the *x* and *y* fields.
max_aspect	Specifies the maximum of the range of aspect ratios the application prefers. This field is expressed as a ratio of the *x* and *y* fields.
base_width	Defines an arithmetic progression, when used with the *width_inc* field, of the preferred window width.
base_height	Defines an arithmetic progression, when used with the *height_inc* field, of the preferred window height.

Related Information

The **XGetNormalHints** subroutine, **XGetSizeHints** subroutine, **XGetZoomHints** subroutine, **XSetNormalHints** subroutine, **XSetSizeHints** subroutine, **XSetZoomHints** subroutine.

XIconSize Data Structure

```
typedef struct {
   int min_width, min_height;
   int max_width, max_height;
   int width_inc, height_inc;
} XIconSize;
```

The **XIconSize** data structure includes the following fields:

min_width	Specifies the minimum icon width.
min_height	Specifies the minimum icon height.
max_width	Specifies the maximum icon width.
max_height	Specifies the maximum icon height.
width_inc	Specifies an arithmetic progression of sizes, from minimum to maximum, that represent the supported icon sizes.
height_inc	Specifies an arithmetic progression of sizes, from minimum to maximum, that represent the supported icon sizes.

Related Information

The **XGetIconSizes** subroutine, **XSetIconSizes** subroutine.

XClassHint Data Structure

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

The **XClassHint** data structure includes the following fields:

res_name	Specifies the application name.
res_class	Specifies the application class.

Related Information

The **XGetClassHint** subroutine, **XSetClassHint** subroutine.

XrmValue Data Structure

```
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 *XrmDat
abase;
```

Database values consist of a size, an address, and a representation type. The representation type allows storage of data tagged by some application defined type (for example, font or color). It has nothing to do with the C language data type or with its class.

The **XrmValue** data structure has the following fields:

size Specifies the size of the resource database, specified in bytes.

addr Specifies the location of the resource database.

XrmOptionDescList Data Structure

```
typedef enum {
   XrmoptionNoArg,            /* Value is specified in
                                 OptionDescRec.value */
   XrmoptionIsArg,            /* Value is the option string
                                 itself */
   XrmoptionStickyArg,        /* Value is characters immediately
                                 following option */
   XrmoptionSepArg,           /* Value is next argument in argv */
   XrmoptionResArg,           /* Resource and value in 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 specification string in
                                 argv */
   char *resourceName;        /* Binding and resource name
                                 (sans application name)
    XrmOptionKind argKind;     /* Which style of option it is */
   caddr_t value;             /* Value to provide if
                                 XrmoptionNoArg */
} XrmOptionDescRec, *XrmOptionDescList;
```

The **XrmOptionDescList** data structure includes the following fields:

option Specifies the option specification string in the *argv* field.

resourceName Specifies the binding and resource name (without the application name).

argKind Specifies the style of option. This field can be one of the following values:

XrmoptionNoArg The value is specified in the *value* field.

XrmoptionIsArg The value is the option string itself.

XrmoptionStickyArg The value is found in the characters immediately following the option.

XrmoptionSepArg The value is the next argument in the *argv* field.

	XrmoptionResArg	The resource and value in the next argument in the *argv* field.
	XrmoptionSkipArg	Ignore this option and the next argument in the *argv* field.
	XrmoptionSkipLine	Ignore this option and the rest of the *argv* field.
value	The value to provide if the *argKind* field is **XrmoptionNoArg.**	

Related Information

The **XrmParseCommand** subroutine.

XAIXDeviceMappingEvent Data Structure

```
typedef struct {
   int type;                /* Event type */
   unsigned long serial     /* Number of last request processed by
                             server */
   Bool send_event;         /* True if from SendEvent request */
   Display *display;        /* Display event was read from */
   Window window;           /* unused */
   int request;             /* AIXMappingDial or AIXMappingLpfk */
   int lpfkmask;            /* lpfk input */
   int lightmask;           /* lpfk output */
   int dialmask;            /* dial mask */
} XAIXDeviceMappingEvent;
```

type Specifies the event type, which is **AIXDeviceMappingNotify**.

serial Specifies the serial number of last event processed in the server.

send_event Specifies if the event was generated by a **SendEvent** protocol request. If it was, the *send_event* field is set to the value of **True**.

display Specifies the connection to the X Server.

window Unused in this request.

lpfkmask Set to new lpfkmask value if the request is **AIXMappingLpfk**.

lightmask Set to the new lightmask value if the request is **AIXMappingLpfk**.

dialmask Set to the new dialmask value if the request is **AIXMappingDial**.

Appendix B. Enhanced X-Windows Toolkit Data Structures

Related Information

The **ApplicationShellClassRec** data structure.
The **ApplicationShellPart** data structure.
The **ApplicationShellWidget** data structure.
The **CompositeClassPart** data structure.
The **CompositePart** data structure.
The **ConstraintClassPart** data structure.
The **ConstraintPart** data structure.
The **CoreClassPart** data structure.
The **CorePart** data structure.
The **OverrideShellClassRec** data structure.
The **OverrideShellPart** data structure.
The **OverrideShellWidget** data structure.
The **ShellClassRec** data structure.
The **ShellPart** data structure.
The **ShellWidget** data structure.
The **TopLevelShellClassRec** data structure.
The **TopLevelShellPart** data structure.
The **TopLevelShellWidget** data structure.
The **TransientShellClassRec** data structure.
The **TransientShellPart** data structure.
The **TransientShellWidget** data structure.
The **VendorShellClassRec** data structure.
The **VendorShellPart** data structure.
The **VendorShellWidget** data structure.
The **WMShellClassRec** data structure.
The **WMShellPart** data structure.
The **WMShellWidget** data structure.
The **XrmValue** data structure.
The **XtAcceptFocusProc** data type.
The **XtActionProc** procedure pointer.
The **XtActionList** data structure.
The **XtAddressMode** enumerated type.
The **XtAlmostProc** data type.
The **ArgList** data structure.
The **XtArgsFunc** data type.
The **XtArgsProc** data type.
The **XtCallbackList** data structure.
The **XtCallbackProc** data type.
The **XtCaseProc** data type.
The **XtConvertArgRec** data structure.
The **XtConvertSelectionProc** data type.
The **XtConverter** data type.
The **XtErrorHandler** data type.
The **XtErrorMsgHandler** data type.
The **XtEventHandler** data type.
The **XtExposeProc** data type.
The **XtGeometryHandler** data type.
The **XtGeometryResult** data structure.

The **XtInitProc** data type.
The **XtInputCallbackProc** data type.
The **XtKeyProc** data type.
The **XtLoseSelectionProc** data type.
The **XtOrderProc** data type.
The **XtPopdownID** data structure.
The **XtProc** data type.
The **XtRealizeProc** data type.
The **XtResource** data structure.
The **XtResourceDefaultProc** data type.
The **XtSelectionCallbackProc** data type.
The **XtSelectionDoneProc** data type.
The **XtStringProc** data type.
The **XtTimerCallbackProc** procedure.
The **XtWidgetClassProc** data type.
The **XtWidgetGeometry** data structure.
The **XtWidgetProc** data type.
The **XtWorkProc** data structure.

CoreClassPart Data Structure

The common fields for all widget classes are defined in the **CoreClassPart** data structure:

```
typedef struct {
   WidgetClass superclass;
    String class_name;
   Cardinal widget_size;
   XtProc class_initialize;
   XtWidgetClassProc class_part_initialize;
   Boolean class_inited;
   XtInitProc initialize;
   XtArgsProc initialize_hook;
   XtRealizeProc realize;
   XtActionList actions;
   Cardinal num_actions;
   XtResourceList resources;
   Cardinal num_resources;
   XrmClass xrm_class;
   Boolean compress_motion;
   Boolean compress_exposure;
   Boolean compress_enterleave;
   Boolean visible_interest;
    XtWidgetProc destroy;
    XtWidgetProc resize;
   XtExposeProc expose;
   XtSetValuesFunc set_values;
   XtArgsFunc set_values_hook;
   XtAlmostProc set_values_almost;
   XtArgsProc get_values_hook;
   XtAcceptFocusProc accept_focus;
   XtVersionType version;
   _XtOffsetList callback_private;
   String tm_table;
   XtGeometryHandler query_geometry;
   XtStringProc display_accelerator;
   caddr_t extension;
} CoreClassPart;
```

Related Information

The **XtArgsFunc** data type.

CorePart Data Structure

The common fields for all widget instances are defined in the **CorePart** structure:

```
typedef struct _CorePart {
   Widget self;
   WidgetClass widget_class;
   Widget parent;
   XrmName xrm_name;
   Boolean being_destroyed;
   XtCallbackList destroy_callbacks;
   caddr_t constraints;
   Position x;
   Position y;
   Dimension width;
   Dimension height;
   Dimension border_width;
   Boolean managed;
   Boolean sensitive;
   Boolean ancestor_sensitive;
   XtEventTable event_table;
   XtTMRec tm;
   XtTranslations accelerators;
   Pixel border_pixel;
   Pixmap border_pixmap;
    WidgetList popup_list;
    Cardinal num_popups;
    String name;
    Screen *screen;
    Colormap colormap;
    Window window;
    Cardinal depth;
    Pixel background_pixel;
    Pixmap background_pixmap;
    Boolean visible;
    Boolean mapped_when_managed;
} CorePart;
```

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** data structure are:

Field	Default Value
self	Address of the widget structure (may not be changed)
widget_class	*widget_class* argument to the **XtCreateWidget** subroutine (may not be changed)
parent	*parent* argument to the **XtCreateWidget** subroutine (may not be changed)
xrm_name	Encoded *name* argument to the **XtCreateWidget** subroutine (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* fields 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	The *name* argument to the **XtCreateWidget** subroutine (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

CompositeClassPart Data Structure

In addition to the **Core** widget class fields, **Composite** widgets have the following class fields:

```
typedef struct {
    XtGeometryHandler geometry_manager;
    XtWidgetProc change_managed;
    XtWidgetProc insert_child;
    XtWidgetProc delete_child;
    caddr_t extension;
} CompositeClassPart;
```

Related Information

The **CompositePart** data structure.

CompositePart Data Structure

In addition to the **CorePart** fields, **Composite** widgets have the following fields defined in the **CompositePart** structure:

```
typedef struct {
   WidgetList children;
   Cardinal num_children;
   Cardinal num_slots;
   XtOrderProc insert_position;
} CompositePart;
```

The default values are filled in by the **Composite** resource list and the **Composite** initialize procedure. The default values for the **CompositePart** data structure are:

Field	Default Value
children	NULL
num_children	0
num_slots	0
insert_position	Internal function **InsertAtEnd**

Related Information

The **CompositeClassPart** data structure.

ConstraintClassPart Data Structure

In addition to the **Composite** class fields, **Constraint** widgets have the following class fields:

```
typedef struct {
    XtResourceList resources;
   Cardinal num_resources;
   Cardinal constraint_size;
   XtInitProc initialize;
   XtWidgetProc destroy;
   XtSetValuesFunc set_values;
   caddr_t extension;
} ConstraintClassPart;
```

Related Information

The **ConstraintPart** data structure.

ConstraintPart Data Structure

In addition to the **CompositePart** fields, **Constraint** widgets have the following fields defined in the **ConstraintPart** data structure:

```
typedef struct { int empty; } ConstraintPart;
```

Related Information

The **ConstraintClassPart** data structure.

ArgList Data Structure

```
typedef something ArgList;

typedef struct {
   String name;
   ArgList value;
} Arg, *ArgList;
```

The **ArgList** data structure is a pointer to the **Arg** data structure. The **ArgList** data type is a C language type which is large enough to contain the following: **caddr_t, char*, long, int*,** or a pointer to a function.

Many of the Intrinsics routines need to receive pairs of resource names and values, called an argument list. These are passed as an **ArgList** structure.

name Specifies the name of the resource.

value Contains the resource value if the size of the resource is less than or equal `ArgList`to the size of an **ArgList** structure. Otherwise, the *value* field is a pointer to the resource value.

Related Information

The **XtMergeArgLists** subroutine, **XtSetArg** subroutine.

XtInitProc Data Type

```
typedef void (*XtInitProc)(Widget, Widget)
   Widget request;
   Widget new;
```

The **XtInitProc** procedure is the **initialize** procedure (the procedure specified in the *initialize* field of a widget class) pointer type for a widget class.

An initialization procedure performs the following:

- Allocates space for and copies any resources that are referenced by address. For example, if a widget has a field that is a **String**, it cannot depend on the characters at that address remaining constant but must dynamically allocate space for the string and copy it to the new space. (note that you should not allocate space for or copy callback lists.)
- Computes values for unspecified resource fields. for example, if the width and height are 0, the widget should compute an appropriate width and height based on ohter resources. This is the only time that a widget should ever directly assign its own width and height.
- Computes values for uninitialized nonresource fields that are derived from resource fields. For example, graphics contexts (GCs) that the widget uses are derived from resources like background, foreground, and font.

An initialization procedure can also check certain fields for internal consistancy. For example, it makes no sense to specify a color map for a depth that does not support that color map.

A subclass compares the difference between a specified size and a size computed by a superclass by checking the *request* and *new* fields in the **XtInitProc** data type.

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 nonresource, that are allowed. A subclass initialize procedure compares the values to resolve potential conflicts.

XtArgsProc Data Type

```
typedef void (*XtArgsProc)(Widget, ArgList, Cardinal*);
   Widget w;
   ArgList args;
   Cardinal *num_args;
```

Description

The **XtArgsProc** specifies the interface for the **initialize_hook** and **get_values_hook** procedures of a widget.

If the **XtArgsProc** procedure is not the value of **NULL**, it is called immediately after the corresponding initialize procedure or in place of it, if the initialize procedure is the value of **NULL**..

Parameters

w	Specifies the widget ID.
args	Specifies the argument list to override the resource defaults.
num_args	Specifies the number of arguments in the argument list.

XtCallbackProc Data Type

```
typedef void (*XtCallbackProc)(Widget, caddr_t, caddr_t);
   Widget w;
   caddr_t client_data;
   caddr_t call_data;
```

The **XtCallbackProc** data type specifies the interface of a callback procedure to be used in callback lists. A client can register the callback and client-specific data (such as a pointer to additional information about the widget) in the *client_data* field. The client data should be the value of **NULL** if all necessary information is in the widget.

The *call_data* field allows the client application to specify data about the callback. This parameter helps the client avoid using the **XtGetValues** subroutine or a widget-specific subroutine to retrieve data from the widget. For example, if the Scrollbar executes the **thumbChanged** callback list, the *call_data* field passes the new position of the thumb.

Note: Do not use the *call_data* field for complex information.

Fields

w	Specifies the widget ID for which the callback is registered.
client_data	Specifies the data that should be passed to the client when the widget executes the client's callback. If all necessary information is in the widget, this parameter has the value of **Null**.
call_data	Specifies any callback data that should be passed to the client when the widget executes the client's callback.

Related Information

The **XtCallbackList** data structure.

The **XtCreateWidget** subroutine, **XtGetValues** subroutine.

XtCallbackList Data Structure

```
typedef struct {
   XtCallbackProc callback;
   caddr_t closure;
} XtCallbackRec, *XtCallbackList;
```

The **XtCallbackList** structure specifies the address of a null-terminated list, and is used to pass callback information to the **XtCreateWidget** subroutine, **XtSetValues** subroutine, or **XtGetValues** subroutine.

callback	Specifies a pointer to the callback procedure.
closure	Specifies any client data to be passed to the callback procedure.

Related Information

The **XtCreateWidget** subroutine, **XtGetValues** subroutine, **XtSetValues** subroutine.

XtWidgetProc Data Type

The **XtWidgetProc** data type specifies the interface for a user defined *destroy*, *resize*, *change_managed*, *insert_child*, or *delete_child* procedures of a widget. These procedures are stored in the following fields of a widget:

destroy

Destroy procedures are called in subclass-to-superclass order. Therefore, a widget's destroy procedure should only deallocate storage specific to the subclass and should not bother with the storage allocated by any of its superclasses. The destroy procedure should only deallocate resources that have been explicitly created by the subclass. Any resource that was obtained from the resource database or was passed in with an argument list was not created by the widget and, therefore, should not be destroyed by it. If a widget does not need to deallocate any storage, the destroy procedure entry in its widget class record can be the value of **NULL**.

resize

If the composite widget wishes to change the size or border width of any of its children, it calls the **XtResizeWidget** subroutine, which first updates the **Core** fields and then calls the **XConfigureWindow** subroutine if the widget is realized.

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 lay out their displayed data again to match the new size. When a parent resizes a child, it calls **XtResizeWidget**, which updates the geometry fields in the widget., configures the window if the widget is realized, and calls the child's resize procedure to notify the child. The resize procedure pointer is of type **XtWidgetProc**.

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 is an unusual case and should occur only for widgets with very trivial 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 new values. The resize procedure should recalculate the layout of internal data as needed. (For example, a centered Label in a window that changes size should recalculate the starting position of the text.) The widget must obey resize as a command and must not treat it as a request. A widget must not issue an **XtMakeGeometryRequest** or **XtMakeResizeRequest** call from its resize procedure.

change_managed

Child widgets are added to and removed from the managed set by using the **XtManageChild**, **XtManageChildren**, **XtUnmanageChild**, and **XtUnmanageChildren** subroutines, which notify the parent to recalculate the physical layout of its children by calling the parent's *change_managed* procedure. The **XtCreateManagedWidget** convenience subroutine calls the **XtCreateWidget** and **XtManageChild** subroutines on the result.

insert_child

To add a child to the parent's list of children, the **XtCreateWidget** subroutine calls the parent's class routine *insert_child*.

Most composite widgets inherit their superclass' operation. The *insert_child* routine calls the *insert_position* procedure and inserts the child at the specified position.

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 (the *num_children* field of the widget == the *num_slots* field), the *insert_child* procedure must first reallocate the array and update the *num_slots* field of the widget. The *insert_child* procedure then places the child wherever it wants and increments the *num_children* field of the widget.

delete_child

Most widgets inherit the *delete_child* procedure from their superclass. Composite widgets that create companion widgets define their own *delete_child* procedure to remove these companion widgets. To remove the child from the parent's children array, the **XtDestroyWidget** function eventually causes a call to the composite parent's *delete_child* procedure.

The **XtWidgetProc** data type contains the following field:

w

Specifies the widget.

Related Information

The **CompositeClassPart** data structure, **ConstraintClassPart** data structure.

The **XFreeGC** subroutine, **XFreePixmap** subroutine, **XtAddEventHandler**subroutine, **XtAppAddTimeout** subroutine, **XtCalloc** subroutine, **XtDestroyWidget** subroutine, **XtFree** subroutine, **XtGetGC** subroutine, **XtMalloc** subroutine, **XtRemoveEventHandler** subroutine, **XtRemoveTimeout** subroutine, **XtResizeWidget** subroutine.

XtOrderProc Data Type

```
typedef Cardinal (*XtOrderProc)(Widget);
   Widget w;
```

The **XtOrderProc** data type is the interface definition for the **insert_position** procedure in a composite widget instance. This procedure is useful when composite widgets need a specific order for their children widgets; it determines where a new child should go in the children list of the widget. This procedure is called from the **insert_child** procedure of the widget class.

The return value of the **insert_position** procedure indicates how many children should go before the widget

w Specifies the widget.

Return Values

0 Indicates that the widget should go before all other children.

num_children Indicates that the widget should go after all other children.

Related Information

The **CompositePart** data structure.

Enhanced X-Windows ShellClassRec Data Structure

```
typedef struct _ShellClassRec {
   CoreClassPart core_class;
   CompositeClassPart composite_class;
   ShellClassPart shell_class;
} ShellClassRec;
```

Related Information

The **ShellPart** data structure.
The **ShellWidget** data structure.

OverrideShellClassRec Data Structure

```
typedef struct _OverrideShellClassRec {
   CoreClassPart core_class;
   CompositeClassPart composite_class;
   ShellClassPart shell_class;
   OverrideShellClassPart override_shell_class;
} OverrideShellClassRec;
```

Related Information

The **OverrideShellPart** data structure.
The **OverrideShellWidget** data structure.

WMShellClassRec Data Structure

```
typedef struct _WMShellClassRec {
   CoreClassPart core_class;
   CompositeClassPart composite_class;
   ShellClassPart shell_class;
   WMShellClassPart wm_shell_class;
} WMShellClassRec;
```

Related Information

The **WMShellPart** data structure.
The **WMShellWidget** data structure.

VendorShellClassRec Data Structure

```
typedef struct _VendorShellClassRec {
   CoreClassPart core_class;
   CompositeClassPart composite_class;
   ShellClassPart shell_class;
   WMShellClassPart wm_shell_class;
   VendorShellClassPart vendor_shell_class;
} VendorShellClassRec;
```

Related Information

The **VendorShellPart** data structure.
The **VendorShellWidget** data structure.

TransientShellClassRec Data Structure

```
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;
```

Related Information

The **TransientShellPart** data structure.
The **TransientShellWidget** data structure.

TopLevelShellClassRec Data Structure

```
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;
```

Related Information

The **TopLevelShellPart** data structure.
The **TopLevelShellWidget** data structure.

ApplicationShellClassRec Data Structure

```
typedef struct _ApplicationShellClassRec {
   CoreClassPart core_class;
   CompositeClassPart composite_class;
   ShellClassPart shell_class;
   WMShellClassPart wm_shell_class;
   VendorShellClassPart vendor_shell_class;
   TopLevelShellClassPart top_level_shell_class;
   ApplicationShellClassPart application_shell_class;
} ApplicationShellClassRec;
```

Related Information

The **ApplicationShellPart** data structure.
The **ApplicationShellWidget** data structure.

ShellPart Data Structure

```
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;
```

allow_shell_resize — 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 the value **XtGeometryNo**. The default value for the *allow_shell_resize* field is **False**.

client_specified — By default, the *client_specified* field is used internally.

create_popup_child_proc — The procedure defined by this field is called by the **XtPopup** subroutine. The default value for the *create_popup_child_proc* field is **NULL**.

geometry — Specifies size and position and is usually done only from a command line or a defaults file. The default value for the *geometry* field is **NULL.**

grab_kind — By default, the *grab_kind* field is used internally.

override_redirect — Setting the *override_redirect* field determines whether or not the shell window is visible to the window manager. If it is the value of **True**, the window is immediately mapped without the intervention of the manager. The default value for the *override_redirect* field is **True** for the **OverrideShell**, and **False** otherwise.

popdown_callback — This field is called during the **XtPopdown** subroutine. The default value for the *popdown_callback* field is **NULL**.

popped_up — By default, the *popped_up* field is used internally.

popup_callback — This is called during the **XtPopup** subroutine. The default value for the *popup_callback* field is **NULL**.

save_under — Setting the *save_under* field instructs the server to attempt to save the contents of windows obscured by the shell when it is mapped and to restore its contents automaticlly later. It can be useful for pop-up menus. The default value for the *save_under* field is **True** for the **OverrideShell** and **TransientShell** widget classes, and **False** otherwise.

spring_loaded — By default, the *spring_loaded* field is used internally.

Related Information

The **ShellClassRec** data structure.
The **ShellWidget** data structure.

OverrideShellPart Data Structure

```
typedef struct { int empty; } OverrideShellPart;
```

Related Information

The **OverrideShellClassRec** data structure.
The **OverrideShellWidget** data structure.

WMShellPart Data Structure

```
typedef struct {
   String title;
   int wm_timeout;
   Boolean wait_for_wm;
   Boolean transient;
   XSizeHints size_hints;
   XWMHints wm_hints;
} WMShellPart;
```

The common shell fields and their default values in the **WMShell** widget class and its subclasses are:

size_hints Specifies resources for sizing the window. This can include the following:

- *max_height* The default value for the *max_height* field is **None**.
- *max_width* The default value for the *max_width* field is **None**.
- *min_height* The default value for the *min_height* field is **None**.
- *min_width* The default value for the *min_width* field is **None**.
- *height_inc* The default value for the *height_inc* field is **None**.
- *width_inc* The default value for the *width_inc* field is **None**.

title This value is a string displayed by the window manager. The default value for the *title* field is the icon name if one is specified. Otherwise it is the name of the application.

transient The default value for the *transient* field is **True** for the **TransientShell** widget class, and **False** otherwise.

wait_for_wm The default value for the *wait_for_wm* field 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 the *wm_timeout* field. When the field is the value of **False**, the shell does not wait for confirmation but relies on asynchronous notification.

wm_hints Specifies resources for window manager hints. This can include the following:

- *min_aspect_x* The default value for the *min_aspect_x* field is **None**.
- *min_aspect_y* The default value for the *min_aspect_y* field is **None**.
- *max_aspect_x* The default value for the *max_aspect_x* field is **None**.
- *max_aspect_y* The default value for the *max_aspect_y* field is **None**.
- *input* The default value for the *input* field is **False**.
- *initial_state* The default value for the *initial_state* field is normal.
- *icon_pixmap* The default value for the *icon_pixmap* field is **None**.

icon_window The default value for the *icon_window* field is **None**.

icon_x The default value for the *icon_x* field is **None**.

icon_y The default value for the *icon_y* field is **None**.

icon_mask The default value for the *icon_mask* field is **None**.

window_group The default value for the *window_group* field is **None**.

wm_timeout 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 the *wait_for_wm* field to the value of **False**. The default value for the *wm_timeout* field is five seconds.

Related Information

The **WMShellClassRec** data structure.
The **WMShellWidget** data structure.

VendorShellPart Data Structure

```
typedef struct {
   int vendor_specific;
} VendorShellPart;
```

Related Information

The **VendorShellClassRec** data structure.
The **VendorShellWidget** data structure.

TransientShellPart Data Structure

```
typedef struct { int empty; } TransientShellPart;
```

Related Information

The **TransientShellClassRec** data structure.
The **TransientShellWidget** data structure.

TopLevelShellPart Data Structure

```
typedef struct {
   String icon_name;
   Boolean iconic;
} TopLevelShellPart;
```

The common shell fields and their default values for the **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.

Related Information

The **TopLevelShellClassRec** data structure.
The **TopLevelShellWidget** data structure.

ApplicationShellPart Data Structure

```
typedef struct {
   char *class;
   XrmClass xrm_class;
   int argc;
   char **argv;
} ApplicationShellPart;
```

The common shell fields and their default values for **Application** shells are:

argc This field is used to initialize the WM_COMMAND standard property. The default value for *argc* is 0 (zero).

argv This field is used to initialize the WM_COMMAND standard property. The default value for *argv* is **NULL**.

Related Information

The **ApplicationShellClassRec** data structure.
The **ApplicationShellWidget** data structure.

ShellWidget Data Structure

```
typedef struct {
   CorePart core;
   CompositePart composite;
   ShellPart shell;
} ShellRec, *ShellWidget;
```

Related Information

The **ShellClassRec** data structure.
The **ShellPart** data structure.

OverrideShellWidget Data Structure

```
typedef struct {
   CorePart core;
   CompositePart composite;
   ShellPart shell;
   OverrideShellPart override;
} OverrideShellRec, *OverrideShellWidget;
```

Related Information

The **OverrideShellClassRec** data structure.
The **OverrideShellPart** data structure.

WMShellWidget Data Structure

```
typedef struct {
   CorePart core;
   CompositePart composite;
   ShellPart shell;
   WMShellPart wm;
} WMShellRec, *WMShellWidget;
```

Related Information

The **WMShellClassRec** data structure.
The **WMShellPart** data structure.

VendorShellWidget Data Structure

```
typedef struct {
   CorePart core;
   CompositePart composite;
   ShellPart shell;
   WMShellPart wm;
   VendorShellPart vendor;
} VendorShellRec, *VendorShellWidget;
```

Related Information

The **VendorShellClassRec** data structure.
The **VendorShellPart** data structure.

TransientShellWidget Data Structure

```
typedef struct {
   CorePart core;
   CompositePart composite;
   ShellPart shell;
   WMShellPart wm;
   VendorShellPart vendor;
   TransientShellPart transient;
} TransientShellRec, *TransientShellWidget;
```

Related Information

The **TransientShellClassRec** data structure.
The **TransientShellPart** data structure.

TopLevelShellWidget Data Structure

```
typedef struct {
   CorePart core;
   CompositePart composite;
   ShellPart shell;
   WMShellPart wm;
   VendorShellPart vendor;
   TopLevelShellPart topLevel;
} TopLevelShellRec, *TopLevelShellWidget;
```

Related Information

The **TopLevelShellClassRec** data structure.
The **TopLevelShellPart** data structure.

ApplicationShellWidget Data Structure

```
typedef struct {
   CorePart core;
   CompositePart composite;
   ShellPart shell;
   WMShellPart wm;
   VendorShellPart vendor;
   TopLevelShellPart topLevel;
   ApplicationShellPart application;
} ApplicationShellRec, *ApplicationShellWidget;
```

Related Information

The **ApplicationShellClassRec** data structure.
The **ApplicationShellPart** data structure.

XtWorkProc Data Type

```
typedef Boolean(*XtWorkProc)(caddr_t)
   caddr_t client_data;
```

The **XtWorkProc** data type specifies the interface definition for user defined idle–time work procedures. The user defined procedure must return the value of **True** if it is done (that is, the work is complete).

client_data Specifies the client data generated when the work procedure was registered.

Related Information

The **XtAppAddWorkProc** subroutine.

XtExposeProc Data Type

Syntax

```
typedef void (*XtExposeProc)(Widget, XEvent *, Region)
   Widget w;
   XEvent *event;
   Region region;
```

Description

The **XtExposeProc** data type specifies the interface definition for user defined **expose** procedure in widget classes. The expose procedure redisplays a widget upon exposure (This redisplay is the responsibility of the widget.). If a widget has no display semantics, specify the value of **NULL** for its **expose** procedure. For example, many composite widgets serve as containers for their children only and have no expose procedure.

Note: If the **XtExposeProc** procedure is the value of **NULL**, the **XtRealizeWidget** subroutine fills in the default bit gravity of **NorthWestGravity** before it calls the widget realize procedure.

Fields

event Specifies the exposure event that identifies the rectangle that requires redisplaying. This parameter contains the bounding box for the *Region* parameter, if the *compress_exposure* field of the widget is the value of **True**.

region Specifies the union of all rectangles in this exposure sequence. This parameter is the value of **NULL** if the *compress_exposure* field of the widget is the value of **False**.

w Specifies the ID of the widget instance that requires displaying.

Related Information

The **XtRealizeWidget** subroutine.

XtEventHandler Data Type

```
typedef void (*XtEventHandler)(Widget, caddr_t, XEvent*);
   Widget w;
   caddr_t client_data;
   XEvent *event;
```

The **XtEventHandler** data type is the interface definition for user defined event handler procedures for widgets that must use event handlers explicitly. Most widgets use the translation manager, instead of using event handlers explicitly.

client_data Specifies the client–specific information registered with the event handler. If the event handler is registered by the widget, this parameter is the value of **NULL**.

event Specifies the triggering event.

w Specifies the widget ID for which to handle events.

XtWidgetGeometry Data Structure

The **XtWidgetGeometry** data 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 the **<X11/X.h>** header file:

```
#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)
```

The **XtCWQueryOnly** value 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.

The **XtMakeGeometryRequest** subroutine (like the corresponding **Xlib XConfigureWindow** subroutine) uses *request_mode* to determine which fields in the **XtWidgetGeometry** structure you want to specify.

The *stack_mode* values are defined in the **<X11/X.h>** header file:

```
#define Above                   0

#define Below                   1

#define TopIf                   2

#define BottomIf                3

#define Opposite                4
```

The Intrinsics also support the following value:

```
#define XtSMDontChange          5
```

The **XtSMDontChange** value indicates that the widget requires its current stacking order preserved.

Related Information

The **XtGeometryResult** data structure.

XtGeometryResult Data Structure

The return codes from geometry managers are:

```
typedef enum _XtGeometryResult {
   XtGeometryYes,
   XtGeometryNo,
   XtGeometryAlmost,
   XtGeometryDone,
} XtGeometryResult;
```

Related Information

The **XtWidgetGeometry** data structure.

XtGeometryHandler Data Type

Syntax

```
typedef XtGeometryResult(*XtGeometryHandler)(Widget,
                                             XtWidgetGeometry*,
                                             XtWidgetGeometry*)
   Widget w;
   XtWidgetGeometry *request;
   XtWidgetGeometry *geometry_return;
```

Description

The **XtGeometryHandler** data type specifies the interface definition for the **geometry_manager** procedure in a composite widget. A class can inherit the geometry manager of its superclass during class initialization.

The same definition is also used for the **query_geometry** procedure in a widget. The **query_geometry** procedure:

- Examines the bits set in the *request->request_mode* parameter.
- Evaluates the preferred geometry of the widget.
- Stores the result in the *geometry_return* parameter. It sets the bits in the *geometry_return->request_mode* parameter to the corresponding geometry fields.
- Generates the appropriate return value.

A bit set to the value of **0** in the mask field of the request means that the child widget does not care about the value of the corresponding field. Then the geometry manager can change it as it wishes. A bit set to **1** means that the child wants that geometry element changed to the value in the corresponding field.

If the geometry manager can satisfy all changes requested and if **XtCWQueryOnly** is not specified, it updates the widget's x, y, width, height, and border_width values appropriately. Then it returns **XtGeometryYes**, and the value of the geometry_return argument is undefined. The widget's window is moved and resized automatically by **XtMakeGeometryRequest**.

Homogeneous composite widgets often find it convenient to treat the widget making the request the same as any other widget, possibly reconfiguring it as part of its layout process, unless **XtCWQueryOnly** is specified. If it does this, it should return **XtGeometryDone** to inform **XtMakeGeometryRequest** that it does not need to do the configuration itself.

Although **XtMakeGeometryRequest** resizes the widget's window (if the geometry manager returns **XtGeometryYes**), it does not call the widget class's resize procedure. The requesting widget must perform whatever resizing calculations are needed explicitly.

If the geometry manager chooses to disallow the request, the widget cannot change its geometry. The value of the geometry_return parameter is undefined, and the geometry manager returns **XtGeometryNo**.

Sometimes the geometry manager cannot satisfy the request exactly, but may be able to satisfy a similar request. That is, it could satisfy only a subset of the requests or a lesser request. In such cases, the geometry manager fills in the *geometry_return* field with the actual changes it is willing to make, including an appropriate mask, and returns **XtGeometryAlmost**. If a bit in the *geometry_return->request_mode* is **0**, the geometry manager does not change the corresponding value if the *geometry_return* field is used immediately in a new request. If a bit is **1**, the geometry manager does change that element to the corresponding value in the *geometry_return* field. More bits may be set in the *geometry_return* field than in the original request if the geometry manager intends to change other fields should the child accept the compromise.

When the **XtGeometryAlmost** value is returned, the widget must decide if the compromise suggested in the *geometry_return* field is acceptable. If it is, the widget must not change its geometry directly; rather, it must make another call to the **XtMakeGeometryRequest** subroutine.

If the next geometry request from this child uses the *geometry_return* box filled in by an **XtGeometryAlmost** return and if there have been no intervening geometry requests on either its parent or any of its other children, the geometry manager must grant the request, if possible. That is, if the child asks immediately with the returned geometry, it should get an answer of **XtGeometryYes**. However, the user's window manager may affect the final outcome.

To return an **XtGeometryYes** value, the geometry manager frequently rearranges the position of other managed children by calling the **XtMoveWidget** subroutine. However, a few geometry managers may sometimes change the size of other managed children by calling the **XtResizeWidget** or the **XtConfigureWidget** subroutine. If **XtCWQueryOnly** is specified, the geometry manager must return how it would react to this geometry request without actually moving or resizing any widgets.

Geometry managers must not assume that the *request* and *geometry_return* fields point to independent storage. The caller is permitted to use the same field for both, and the geometry manager must allocate its own temporary storage , if necessary.

Fields

geometry_return	Specifies the geometry request returned by the geometry manager.
request	Specifies the request for a geometry change.
w	Specifies the widget.

Return Values

XtGeometryAlmost	Indicates that at least one field in the *PreferredReturn* parameter is different from the corresponding field in the *Intended* parameter or if a bit is set in *PreferredReturn* parameter that is not set in the *Intended* parameter of the **query_geometry** procedure.
XtGeometryNo	Indicates that the preferred geometry is identical to the current geometry.
XtGeometryYes	Indicates that the proposed geometry change is acceptable without modification.

Related Information

The **XtQueryGeometry** subroutine.

XtResource Data Structure

The declaration for the **XtResource** data 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:

resource_name Contains the name used by clients to access the field in the widget. This name starts with a lower–case letter. It is spelled almost the same as the field name, except that underscores (_) are deleted, and the next leter replaced by its uppercase counterpart. 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 **<X11/StringDefs.h>** header file. The Intrinsics symbolic resource names begin with **XtN** and are followed by the string name; for example, **XtNbackgroundPixel** for **backgroundPixel**.

resource_class 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 and bold, but all fonts should have the class **Font**. Changing 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** (for example. XtCBackground).

resource_type

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. The Intrinsics define the following resource types:

Resource Type	Structure or Field Type
XtRAcceleratorTable	XtAccelerators
XtRBoolean	Boolean
XtRBool	Bool
XtRCallback	XtCallbackList
XtRColor	XColor
XtRCursor	Cursor
XtRDimension	Dimension
XtRDisplay	Display*
XtRFile	FILE*
XtRFloat	float
XtRFont	Font
XtRFontStruct	XFontStruct*
XtRFunction	(*)()
XtRInt	int
XtRPixel	Pixel
XtRPixmap	Pixmap
XtRPointer	caddr_t
XtRPosition	Position
XtRShort	short
XtRString	char*
XtRTranslationTable	XtTranslations
XtRUnsignedChar	unsigned char
XtRWidget	Widget
XtRWindow	Window

resource_size

The size of the physical representation in bytes and should be specified as `sizeof(type)` so that the compiler can fill in the value.

resource_offset The offset in bytes of the field within the widget. Use the **XtOffset** macro to retrieve this value.

default_type 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.

default_address 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 (for example, 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** data structure in which to store the result.

Related Information

The **<X11/StringDefs.h>** header file.

The **XrmValue** data structure.

The **XtOffset** macro.

XtResourceDefaultProc Data Type

The **XrmValue** data structure is of **XtResourceDefaultProc** data type, which has the following structure:

```
typedef void (*XtResourceDefaultProc)(Widget, int, XrmValue*)
   Widget widget;
    int offset;
    XrmValue *value;
```

The **XtResourceDefaultProc** data type fills in the *addr* field of the *value* parameter with a pointer to the default data in its correct type.

Note: The *default_address* field in the resource structure is declared as a **caddr_t**. On some machine architectures, this may be insufficient to hold procedure variables.

When the *default_address* field of the **XtResource** data structure contains the resource type of **XtRCallProc**, the **XtResourceDefaultProc** data type is automatically called with the widget, the *resource_offset* field, and a pointer to the **XrmValue** data structure in which to store the result.

The fields of the **XtResourceDefaultProc** data type are as follows:

widget	Specifies the widget whose resource is to be obtained.
offset	Specifies the offset of the field in the widget record.
value	Specifies the resource value to fill in.

Related Information

The **XtResource** data structure.

XrmValue Data Structure

```
typedef struct {
   unsigned int size;
    caddr_t addr;
} XrmValue, *XrmValuePtr;
```

XtConverter Data Type

```
typedef void (*XtConverter)(XrmValue*, Cardinal*,
                XrmValue*, XrmValue*);
    XrmValue *args;
    Cardinal *num_args;
    XrmValue *from;
    XrmValue *to;
```

Description

The **Xtconverter** data type specifies the interface definition for subroutines that convert resources from one type to another.

Type converters do the following actions:

- Check to see that the number of arguments passed is correct
- Attempt the type conversion
- If successful, return a pointer to the data in the to parameter; otherwise, call **XtWarningMsg** and return without modifying the to parameter.

Most type converters just take the data described by the from parameter and return data by writing into the specified to parameter. A few need other information which is available in the specified argument list. A type converter can invoke another type converter, which allows differing sources that may convert into a common intermediate result to make maximum use of the type converter cache.

The address written to–>addr can not be that of a lock variable because this is not valid after the converter returns. It should be a pointer to a static variable.

The **XtConverter** data structure contains the following fields:

args Specifies a list of additional **XrmValue** arguments to the converter if additional context is needed to perform the conversion. Otherwise, this field is the value of **NULL**. For example, the string–to–font converter needs the widget's screen, or the string–to–pixel converter needs the widget's screen and colormap.

num_args Specifies the number of additional **XrmValue** arguments if additional context is needed. Otherwise, this field is the value of **0**.

from Specifies the value to convert.

to Specifies the descriptor to use to return the converted value.

XtAddressMode Enumerated Type

```
typdef enum {
  /*address mode                parameter representation*/
   XtAddress,                   /*address*/
    XtBaseOffset,                  /*offset*/
   XtImmediate,                    /*constant*/
    XtResourceString,           /*resource name string*/
    XtResourceQuark,              /*resource name quark*/
} XtAddressMode;
```

Related Information

The **XtConvertArgRec** data structure.
The **XtAppAddConverter** subroutine.

XtConvertArgRec Data Structure

```
typedef struct {
    XtAddressMode address_mode;
    caddr_t address_id;
    Cardinal size;
} XtConvertArgRec, *XtConvertArgList;
```

address_id Specifies the address of the resource. See the *address_mode* field definition for details.

address_mode Specifies how the *address_id* field should be interpreted. This field may have the following values:

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 a widget base.
XtResourceQuark	Is an internally compiled form of an **XtResourceString**.

size Specifies the length of the data in bytes.

Related Information

The **XtAddressMode** enumerated type.
The **XtAppAddConverter** subroutine.

XtActionList Data Structure

```
typedef struct _XtActionsRec {
   String action_name;
   XtActionProc action_proc;
} XtActionsRec, *XtActionList;
```

action_name Specifies the name used in translation tables to access the procedure.

action_proc Specifies a procedure pointer of type **XtActionProc**, which points to a procedure that implements the functionality.

Related Information

The **XtActionProc** procedure pointer.

XtActionProc Procedure Pointer

```
typedef void (*XtActionProc)(Widget, XEvent*,
            String*, Cardinal*);
   Widget w;
   XEvent *event;
   String *params;
   Cardinal *num_params;
```

The **XtActionProc** procedure pointer specifies the interface definition for action procedures. 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.

w Specifies the widget that caused the action to be called.

event Specifies the event that caused the action to be called. 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 that were specified in the translation table as arguments to the action.

num_params Specifies the number of arguments specified in the translation table.

Related Information

The **XtActionList** data structure.

The **XtAppAddActions** subroutine.

XtKeyProc Data Type

The translation manager provides support for automatically translating key codes in incoming key events into KeySyms. The **XtKeyProc** data type provides the interface definition for Keycode to KeySym translators.

```
typedef void (*XtKeyProc)(Display*, KeyCode, Modifiers,
            Modifiers*, KeySym*);
    Display *display;
    KeyCode keycode;
    Modifiers modifiers;
    Modifiers *modifiers_return;
    KeySym *keysym_return;
```

The **XtKeyProc** data type takes a Keycode and modifiers and produces a KeySym. For a given key translator subroutine, the *modifiers_return* parameter will be a constant that indicates the subset of all modifiers that are examined by the key translator. Applications should register this key converter with the **XtSetKeyTranslator** subroutine.

The following fields are included in the **XtKeyProc** data type:

display	Specifies the display that the key code is from.
keycode	Specifies the key code to translate.
modifiers	Specifies the modifiers to the key code.
modifiers_return	Returns a mask that indicates the subset of all modifiers are examined by the key translator.
keysym_return	Returns the resulting KeySym.

Related Information

The **XtCaseProc** data type.

The **XtRegisterCaseConverter** sburoutine, **XtSetKeyTranslator** subroutine, **XtTranslateKeycode** subroutine.

XtCaseProc Data Type

```
typedef void (*XtCaseProc)(KeySym*, KeySym*, KeySym*);
   KeySym *keysym;
   KeySym *lower_return;
   KeySym *upper_return;
```

The **XtCaseProc** data type specifies the interface for a case converter procedure. It allows capitalization of nonstandard key symbols. A case conversion routine should be registered with the Intrinsics library by using the **XtRegisterCaseConverter** subroutine. The **XtConvertCase** subroutine calls the appropriate user defined **XtCaseProc** *Procedure*. If there is no case distinction, the user defined **XtCaseProc** *Procedure* should store the key symbols in both return values.

keysym Specifies the key symbol for the conversion.

lower_return Specifies the lowercase equivalent for the key symbol.

upper_return Specifies the uppercase equivalent for the key symbol.

Related Information

The **XtConvertCase** subroutine, **XtRegisterCaseConverter** subroutine.

XtAcceptFocusProc Data Type

```
typedef Boolean (*XtAcceptFocusProc)(Widget, Time);
   Widget w;
   Time *time;
```

The **XtAcceptFocusProc** data type defines the user written interface for the *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 even when it does not accept 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** library focus notification mechanism explicitly by specifying translations for the **FocusIn** and **FocusOut** events.

Widgets that do not want the input focus should set the *accept_focus* procedure pointer to the value of **NULL**.

Widgets that need the input focus can call the **XSetInputFocus** subroutine explicitly.

time Specifies the X time of the event causing the *accept_focus* procedure.

w Specifies the widget ID.

Related Information

The **XtCallAcceptFocus** subroutine, **XSetInputFocus** subroutine, **XtSetKeyboardFocus** subroutine.

XtAlmostProc Data Type

```
typedef void (*XtAlmostProc)(Widget, Widget, XtWidgetGeometry*,
              XtWidgetGeometry*);
   Widget w;
   Widget new_widget_return;
   XtWidgetGeometry *request;
   XTWidgetGeometry *reply;
```

The **XtAlmostProc** data type defines the interface for the *set_values_almost* field of a widget. This field is a procedure pointer. Most classes inherit this operation from their superclass by specifying **XtInheritSetValuesAlmost** in the class initialization. The core *set_values_almost* field accepts the compromise suggest.

The *set_values_almost* procedure is called when the geometry manager cannot satisfy a client's request to set the window geometry with the **XtSetValues** subroutine. The geometry manager returns the **XtGeometryAlmost** value with a compromise geometry.

The *set_values_almost* procedure takes the original geometry and the compromise geometry and determines an acceptable compromise, which may be the current compromise or a different compromise. It returns the results in the *new_widget_return* field, which is then sent to the geometry manager for another try.

new_widget_return	Specifies the new widget which will store the geometry changes.
reply	Specifies the compromise geometry returned by the geometry manager.
request	Specifies the original geometry request sent to the geometry manager.
w	Specifies the widget ID on which the geometry change is requested.

XtArgsFunc Data Type

```
typedef Boolean (*XtArgsFunc)(Widget, Arglist, Cardinal*);
   Widget w;
   ArgList args;
   Cardinal *num_args;
```

The **XtArgsFunc** specifies the interface for the *set_values_hook* procedure pointer of a widget. Widgets with a subpart can set the resource values with the **XtSetValues** subroutine and a *set_values_hook* procedure.

args	Specifies the argument list for the **XtCreateWidget** subroutine.
num_args	Specifies the number of arguments in the argument list.
w	Specifies the widget ID whose non–widget resource values are to be changed.

Related Information

The **CoreClassPart** data structure.

The **XtCreateWidget** subroutine, **XtSetValues** subroutine.

XtPopdownID Data Structure

```
typedef struct {
   Widget shell_widget;
   Widget enable_widget;
} XtPopdownIDRec, *XtPopdownID;
```

The **XtPopdownID** structure identifies the widgets involved in the **XtCallbackPopdown** subroutine. The address of an **XtPopdownID** structure is passed as the client data to the **XtCallbackPopdown** subroutine.

enable_widget	Specifies the widget used to pop the (pop-up) shell.
shell_widget	Specifies the pop-up shell to be popped down.

Related Information

The **XtCallbackPopdown** subroutine, **XtSetSensitive** subroutine.

XtConvertSelectionProc Data Type

```
typedef Boolean (*XtConvertSelectionProc)(Widget, Atom*, Atom*,
                 Atom*, caddr_t*, unsigned long*, int*);
   Widget w;
   Atom *selection;
   Atom *target;
   Atom *type_return;
   caddr_t *value_return;
   unsigned long *length_return;
   int *format_return;
```

The **XtConvertSelectionProc** data type is the interface definition for user defined procedures that get the value of a selection as a given type from the current selection owner.

Each **XtConvertSelectionProc** data type should respond to the target value **TARGETS** by returning a value containing the list of the targets that will be used for the selection conversion.

format_return — Specifies a pointer into which the size (in bits) of the data elements of the selection value is to be stored.

length_return — Specifies a pointer into which the number of elements in the *ValueReturn* parameter is to be stored.

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 filename, text, or a window.

type_return — Specifies a pointer to an atom that will store the converted value of the selection. For example, a filename or text could specify **XA_STRING** as the value for this parameter.

value_return — Specifies a pointer into which a pointer to the converted value of the selection is stored. The selection owner is responsible for allocating this storage.

If the selection owner provided an **XtSelectionDoneProc** procedure for the selection, the storage is owned by the selection owner. Otherwise, the storage is owned by the Intrinsics selection mechanism.

w — Specifies the ID of the widget that currently owns the selection.

Return Values

False Indicates that the conversion did not take place. The values of the return parameters are undefined.

True Indicates that the selection owner successfully converted the selection to the target type.

Related Information

The **XtLoseSelectionProc** data type, **XtSelectionDoneProc** data type, **XtSelectionCallbackProc** data type.

The **XtDisownSelection** subroutine, **XtGetSelectionValue** subroutine, **XtGetSelectionValues** subroutine, **XtOwnSelection** subroutine.

XtLoseSelectionProc Data Type

```
typedef void (*XtLoseSelectionProc)(Widget, Atom*);
   Widget w;
   Atom* selection;
```

The **XtLoseSelectionProc** data type specifies the interface definition for user defined procedures that are called by the Intrinsics selection mechanism to inform the specified widget that it has lost ownership of the specified selection. The user defined procedures do not initiate the loss of the selection ownership.

selection Specifies the atom that describes the selection type.

w Specifies the widget that has lost selection ownership.

Related Information

The **XtConvertSelectionProc** data type, **XtSelectionCallbackProc** data type, **XtSelectionDoneProc** data type.

The **XtOwnSelection** subroutine.

XtSelectionCallbackProc Data Type

```
typedef void (*XtSelectionCallbackProc)(Widget, caddr_t, Atom*,
              Atom*, caddr_t, unsigned long*, int*);
   Widget w;
   caddr_t client_data;
   Atom* selection;
   Atom* type;
   caddr_t value;
   unsigned long* length;
   int* format;
```

The **XtSelectionCallbackProc** data type specifies the interface for the user defined procedures that are called by the Intrinsics selection mechanism to deliver the requested selection to the requester.

An **XT_CONVERT_FAIL** atom specified in the *Type* parameter, upon return of the user defined procedure, indicates that the selection conversion failed because the selection owner did not respond within the Intrinsics selection time–out interval.

Parameters

client_data Specifies a value passed in by the widget when the selection was requested.

format Specifies the size in bits of the data elements of value.

length Specifies the number of elements in value.

selection Specifies the type of selection that was requested.

type Specifies the type used to represent the selection value (for example, **XA_STRING**).

value Specifies a pointer to the selection value.

The requesting client owns the storage allocated for this parameter. Use the **XtFree** subroutine to de–allocate the storage space when this routine completes.

w Specifies the widget that requested the selection value.

Related Information

The **XtConvertSelectionProc** data type, **XtLoseSelectionProc** data type, **XtSelectionDoneProc** data type.

The **XtFree** subroutine, **XtGetSelectionValue** subroutine, **XtGetSelectionValues** subroutine.

XtSelectionDoneProc Data Type

```
typedef void (*XtSelectionDoneProc)(Widget, Atom*, Atom*);
   Widget w;
   Atom* selection;
   Atom* target;
```

The **XtSelectionDoneProc** data type specifies the interface definition for user defined procedures that are called by the Intrinsics selection mechanism to inform the selection owner when a selection requester 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 requester.

The selection owner that registers an **XtSelectionDoneProc** also owns the storage containing the converted selection value.

selection Specifies the atom that describes the selection type that was converted.

target Specifies the target type to which the conversion was done.

w Specifies the ID of the widget that owns the converted selection.

Related Information

The **XtConvertSelectionProc** data type, **XtLoseSelectionProc** data type, **XtSelectionCallbackProc** data type.

XtErrorHandler Data Type

```
typedef void (*XtErrorHandler)(String);
   String message;
```

The **XtErrorHandler** data type specifies the interface definition for the low–level error and warning handler procedure. The error handler should display the specified message string in an appropriate manner.

message Specifies the error message.

Related Information

The **XtAppSetErrorHandler** subroutine, **XtAppSetWarningHandler** subroutine.

XtErrorMsgHandler Data Type

```
typedef void (*XtErrorMsgHandler)(String, String, String,
             String, String *, Cardinal *);
   String name;
   String type;
   String class;
   String defaultp;
   String * params;
   Cardinal* num_params;
```

The **XtErrorMsgHandler** data type specifies the interface definition for high–level error and warning handler procedures.

The standard **printf** notation is used to substitute the parameters into the message.

class	Specifies the resource class of the error message.
defaultp	Specifies a default message if an error database entry is not found.
name	Specifies the name that is concatenated with the specified type to form the resource name of the error message. The specified name can be a general error, such as **invalidParameters** or **invalidWindow**.
num_params	Specifies the number of values in the parameter list.
params	Specifies a pointer to a list of values to be substituted in the message.
type	Specifies the type that is concatenated with the name to form the resource name of the error message. The specified type gives extra information.

Related Information

The **XtAppSetErrorMsgHandler** subroutine, **XtAppSetWarningMsgHandler** subroutine.

XtInputCallbackProc Data Type

```
typedef void (*XtInputCallbackProc)(caddr_t, int*, XtInputId*);
   caddr_t client_data;
   int* source;
   XtInputId* id;
```

The **XtInputCallbackProc** data type specifies the interface definition for callback procedures used when there are file events.

client_data Specifies the client data registered for this procedure in the **XtAppAddInput** subroutine.

id Specifies the ID returned from the corresponding **XtAppAddInput** subroutine.

source Specifies the source file descriptor generating the event.

Related Information

The **XtAppAddInput** subroutine, **XtRemoveInput** subroutine.

XtProc Data Type

```
typedef void (*XtProc)();
```

The **XtProc** data type specifies the interface definition for the class initialization procedure pointer type. Most class records can be initialized completely at compile time. In some cases, however, a class may need to register type converters or perform other sorts of one–time initialization.

A widget class indicates that is has no class initialization procedure by specifying the value of **NULL** in its *class_initialize* field.

Related Information

The **XtWidgetClassProc** data type.

XtWidgetClassProc Data Type

```
typedef void (*XtWidgetClassProc)(WidgetClass);
  WidgetClass widget_class;
```

The **XtWidgetClassProc** data type provides the interface definition for the user defined procedures that will be stored in the *class_part_initialize* procedure field of a widget.

Widgets have class initializations will be called exactly once. Some classes need to perform additional initializations for fields in its part of the class record. These are performed not just for the particular class, but subclasses as well.

For classes that do not define new class fields and do not need extra processing, the value of **NULL** should be specified in the *class_part_initialize* field of a widget class.

widget_class Specifies the class of the widget.

Related Information

The **XtProc** data type.

XtRealizeProc Data Type

```
typedef void (*XtRealizeProc)(Widget, XtValueMask*,
             XSetWindowAttributes*);
  Widget w;
  XtValueMask *value_mask;
  XSetWindowAttributes *attributes;
```

The **XtRealizeProc** data type specifies the interface definition for the user defined realize procedure in a widget class.

The generic **XtRealizeWidget** subroutine fills in a mask and a corresponding **XSetWindowAttributes** data structure. It sets the following fields based on information in the **Core** structure:

- The *background_pixmap* field (or the *background_pixel* field if the *background_pixmap* field is NULL) is filled in from the corresponding field.
- The *border_pixmap* field (or the *border_pixel* field if the *border_pixmap* field is NULL) is filled in from the corresponding field.
- The *event_mask* field is filled in based on the event handlers registered, the event translations specified, whether the *expose* field is non-NULL, and whether the *visible_interest* field is **True**.
- The *bit_gravity* field is set to **NorthWestGravity** if the *expose* field is NULL.
- The *do_not_propagate_mask* field is set to propagate all pointer and keyboard events up the window tree. A composite widget can implement functionality caused by an event 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 the corresponding bits in *value_mask*) can be set by the *realize* procedure.

Note that because the *realize* procedure is not a chained operation, the widget class *realize* procedure must update the **XSetWindowAttributes** structure with all the appropriate fields from non-**Core** superclasses.

A widget class can inherit its *realize* procedure from its superclass during class initialization. The *realize* procedure defined for **Core** calls the **XtCreateWindow** subroutine with the passed *ValueMask* and *Attributes* parameters and with the *WindowClass* and *VisualPtr* parameters set to **CopyFromParent**. Both **CompositeWidgetClass** and **ConstraintWidgetClass** inherit this *realize* procedure, and most new widget subclasses can do the same.

The most common noninherited *realize* procedures set the bit_gravity in the mask and the attributes to the appropriate value and then create the window. For example, depending on its justification, the *bit_gravity* field can be set to the value of **WestGravity**, **CenterGravity**, or **EastGravity**. Consequently, shrinking the widget just moves the bits appropriately, and no **Expose** event is needed for repainting.

If the children of a composite widget should be realized in a particular order (typically to control the stacking order), that composit widget should call the **XtRealizeWidget** subroutine on its children in the appropriate order from within its own *realize* procedure.

Widgets that have children and that are not a subclass of **compositeWidgetClass** are responsible for calling the **XtRealizeWidget** subroutine on their children, usually from within the *realize* procedure.

The **XtRealizeProc** data type contains the following fields:

attributes	Specifies the window attributes to use in the **XCreateWindow** subroutine.
value_mask	Specifies the fields from the attributes structure to use.
w	Specifies the ID of the widget.

Related Information

The **XSetWindowAttributes** data structure.

The **XtCreateWindow** subroutine, **XtRealizeWidget** subroutine.

XtSetValuesFunc Data Type

```
typedef Boolean (*XtSetValuesFunc)(Widget, Widget, Widget);
   Widget current;
   Widget request;
   Widget new;
```

The **XtSetValuesFunc** data type specifies the inderface definition for the user defined procedure in the *set_values* field of a widget class. This procedure should recompute any field derived from resources that are changed. 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, you can specify the value of **NULL** for the *set_values* field in the class record.

Like the **initialize** field, the *set_values* field mostly deals with the fields defined in the subclass, but it has to resolve conflicts with its superclass, especially conflicts over width and height. Sometimes it is necessary for a subclass to overwrite values filled in by its superclass, particularly in the case of size calculations.

The *new* and *request* parameters provide information for a subclass to determine the difference between a specified size and a size computed by its superclass. The *request* parameter is the widget as originally requested. The *new* parameter starts with the values of the *request* parameter, but has been modified by all superclass *set_values* fields called so far. A widget does not need to refer to the *request* parameter unless it must resolve conflicts between the widget in the *current* parameter and the widget in the *new* parameter. Any changes, including geometry changes, that the widget needs to make should be made to the widget specified in the *new* parameter.

The *set_values* field must return a Boolean value that indicates if the widget needs to be redisplayed. A change in the geometry fields alone does not require the *set_values* field to return the value of **True**; the X Server will eventually generate an **Expose** event, if necessary. After calling all the *set_values* fields, the **XtSetValues** subroutine forces a redisplay by calling the **XClearArea** subroutine if any of the *set_values* procedures returned the value of **True**. Therefore, a *set_values* field should not do its own redisplaying.

The *set_values* field should not do any work in response to changes in geometry. A widget should do any geometry–related work in its **Resize** procedure.

It is permissible to call **XtSetValues** before a widget is realized. Therefore, the *set_values* field must not assume that the widget is realized.

current — Specifies a copy of the widget as it was before the **XtSetValues** subroutine was called.

request — Specifies a copy of the widget with all values changed as specified in the **XtSetValues** subroutine before any class *set_values* fields have been called.

new — Specifies the widget with the new values that are actually allowed.

Related Information

The **XClearArea** subroutine, **XtSetValues** subroutine.

XtStringProc Data Type

```
typedef void (*XtStringProc)(Widget, String)
   Widget w;
   String* string;
```

The **XtStringProc** data type specifies the interface definition for the user defined procedures stored in the *display_accelerator* field of a widget. Accelerators can be specified 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.

string Specifies the string representation of the accelerators for this widget.

w Specifies the ID of the widget that the accelerators are installed on.

Related Information

The **CoreClassPart** data structure.

The **XtParseAcceleratorTable** subroutine, **XtInstallAccelerators** subroutine, **XtInstallAllAccelerators** subroutine.

XtTimerCallbackProc Procedure

```
typedef void (*XtXtTimerCallbackProc)
            (caddr_t, XtIntervalID*);
   caddr_t client_data;
   XtIntervalId *id;
```

The **XtTimerCallbackProc** procedure specifies the interface definition for the user defined procedures to be used when time-outs expire.

client_data Specifies the client data that was registered for this procedure in the **XtAppAddTimeOut** subroutine.

id Specifies the ID returned from the corresponding **XtAppAddTimeOut** subroutine.

Related Information

The **XtAppAddTimeOut** subroutine.

Appendix C. Enhanced X-Windows Extension Data Structures

The **_XExtCodes** data structure
The **xDoSomethingReq** data structure
The **xResourceReq** data structure
The **XLPFKeyPressedEvent** data structure
The **XDialRotatedEvent** data structure
The **XAIXFocusChangeEvent** data structure

_XExtCodes Data Structure

```
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;
```

XLPFKeyPressedEvent Data Structure

```
typedef struct {
   int type;                /* of event */
   unsigned long serial;    /* number of last request processed by
                               the server */
   Bool send_event;         /* true if this came from a SendEvent
                               request */
   Display *display;        /* display the ivent was read from */
   Window window;           /* "event" window it is reported
                               relative to */
   Window root;             /* root window that the event occurred
                               on */
   Window subwindow;        /* child window */
   Time time;               /* milliseconds */
   int x, y;                /* pointer x, y coordinates in the
                               event window */
   int x_root, y_root;      /* coordinates relative to root */
   unsigned int state;      /* key or button mask */
   unsigned int keycode;    /* detail */
   Bool same_screen;        /* same screen flag */

} XLPFKeyEvent;

typedef XLPFKeyEvent XLPFKeyPressedEvent;
```

XDialRotatedEvent Data Structure

```
typedef struct {
   int type;                  /* of event */
   unsigned long serial;      /* number of last request processed by
                               the server */
   Bool send_event;           /* true if this came from a SendEvent
                               request */
   Display *display;          /* display the ivent was read from */
   Window window;             /* "event" window it is reported
                                 relative to */
   Window root;               /* root window that the event occurred
                               on */
   Window subwindow;          /* child window */
   Time time;                 /* milliseconds */
   int x, y;                  /* pointer x, y coordinates in the
                                 event window */
   int x_root, y_root;        /* coordinates relative to root */
   unsigned int state;        /* key or button mask */
   short int dialval;         /* dial value */
   short int dialnum;         /* dial number */
   Bool same_screen;          /* same screen flag */

} XRotateEvent;

typedef XRotateEvent XDialRotatedEvent;
```

XAIXFocusChangeEvent Data Structure

```
typedef struct {
   int type;                  /* AIXFocusIn or AIXFocusOut */
   unsigned long serial;      /* number of last request processed
                                 by the server */
   Bool send_event;           /* true if this came from a SendEvent
                                 request */
   Display *display;          /* display the ivent was read from */
   Window window;             /* "event" window it is reported
                                 relative to */
   short mode;                /* NotifyNormal */
   short devtype;             /* dial or lpfk */
   int detail;                /* NotifyAncestor, NotifyVirtual,
                                 NotifyInferior, NotifyNonLinear,
                                 NotifyNonLinearVirtual,
                                 NotifyPointer, NotifyPointerRoot,
                                 NotifyDetailNone, */

} XAIXFocusChangeEvent;

typedef XAIXFocusChangeEvent XAIXFocusInEvent;
typedef XAIXFocusChangeEvent XAIXFocusOutEvent;
```

Index

Symbols

Numbers

A

B

C

D

E

F

G

H

I

J

K

L

M

N

O

P

Q

R

S

T

U

V

W

X

Z

Reader's Comment Form

AIX Calls and Subroutines Reference: User Interface for IBM RISC System/6000

SC23–2198–00

Please use this form only to identify publication errors or to request changes in publications. Your comments assist us in improving our publications. Direct any requests for additional publications, technical questions about IBM systems, changes in IBM programming support, and so on, to your IBM representative or to your IBM-approved remarketer. You may use this form to communicate your comments about this publication, its organization, or subject matter, with the understanding that IBM may use or distribute whatever information you supply in any way it believes appropriate without incurring any obligation to you.

☐ If your comment does not need a reply (for example, pointing out a typing error), check this box and do not include your name and address below. If your comment is applicable, we will include it in the next revision of the manual.

☐ If you would like a reply, check this box. Be sure to print your name and address below.

Page	Comments

Please contact your IBM representative or your IBM–approved remarketer to request additional publications.

Please print

Date _____

Your Name _____

Company Name _____

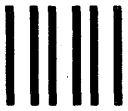
Mailing Address _____

_____

_____

Phone No. ( ) _____
Area Code

No postage necessary if mailed in the U.S.A

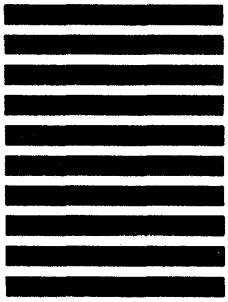


NO POSTAGE
NECESSARY
IF MAILED
IN THE
UNITED STATES

BUSINESS REPLY MAIL

FIRST CLASS PERMIT NO. 40 ARMONK, NEW YORK

POSTAGE WILL BE PAID BY ADDRESSEE



International Business Machines Corporation
Department 997, Building 997
11400 Burnet Rd.
Austin, Texas 78758–3493

Fold

Fold

Cut or Fold Along Line

Fold and Tape

Please Do Not Staple

Fold and Tape

International Business Machines
Corporation
11400 Burnet Road
Austin, Texas 78758–3493

Printed in the
United States of America

SC23-2198-00